Optimization of Assignments for Teaching Assistants at UW

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

December 10, 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. Besides 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:

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

2 Simplification

In order to determine the best assignment that satisfy the above constraints, we will consider two things: what information is needed to find the optimal solution, and what assumption we need to make.

2.1 Input

Below are the information needed to find the optimal solution:

- Preference of candidates 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.
- 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.
- Preference of courses for each candidate to each role, which should also be a numerical value between 0.0 and 1.0 (larger indicates higher preference level).
- Capacity for each course (number of candidates needed for each role of any course, each course may differ).

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.

For this project, we will ignore the need of other roles, and only focus on teaching assistant role.

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 an assignment of candidates to courses, which should significantly consider the preference level of courses and candidates to each other. An assignment can be 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.

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)
    SUBTRACT min(R_A) from R_A
FOR EACH (column C_A in A)
    SUBTRACT 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:
    SUBTRACT min(A) from unlabeled
             R_A
    ADD min(A) to unlabeled C_A
    REPEAT from LABEL
```

For this problem, the original Hungatian Algorithm has to be modified to be compatible with this problem:

1. The result of this problem is a many-to-one matching (multiple candidates assigned to one course). In order to convert the problem to one-to-one matching, we will be splitting each course into slots (for example, if course A requires 5 candidates to be assigned, we will have 5 "slots" from A1 to A5, to corresponds to the 5 candidates wanted by A).
2. Hungarian Algorithm also needs to run on a square matrix. We assume that there will always be more candidates than slots (if not, the department will need to advertise more to get more student apply as candidate). Thus, we can add "dummy" slots to make number of candidates and number of slots equal to each other. If a candidate matches to one of the dummy slots, this candidate is unselected for the row of teaching assistant.
3. A "cost matrix" needs to be constructed for Hungarian Algorithm, and optimal solution (which is the output of Hungarian Algorithm) has minimized total cost. We let the row of cost matrix to be candidates, and column be the slots. Therefore, the value of $(i.j)$ should be:
   - if j is a "dummy" slot, then (i, j) should be 0 regardless of i. We need all "dummy" slots to have the same value, so the optimality if all "non-dummy" matches are not influenced by the dummy variables.
   - if i is not qualified to teach j, the value of (i, j) should be infinity, therefore the Hungarian algorithm will avoid large cost and not choosing the unqualified entry. If the output assignment's cost is infinity, it indicates that no possible matching is available.
   - If i qualified to teach j, the (i, j) entry should be $2 -$ i's preference to j $-$ j's preference to i therefore the cost is smaller if the sum of preference of candidate and course to each other is larger.

After making such modifications, Hungarian's algorithm will output an assignment of candidates to slots, which can be transferred to an assignment of candidates to courses. The sum of preferences to each other for all matched pair of candidates and courses is maximized.

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 Evaluation of methods

5.1 Scoring of assignments

In order to compare and contrast each algorithm discussed above, we will develop a "score function", which takes input of a produced assignment of candidates and courses, and output a numerical score, which higher score indicates better quality of the assignment.
The input of the scoring function should be a serie of binary variables:

$$\sigma_{i,j} = \begin{cases} 1, \text{ if candidate i is assigned to course j} \\ 0, \text{ else} \end{cases}$$

for each candidate i and each course j in the assignment.
Let $m_{i,j}$ be candidate i's preference score to course j, and $n_{j,i}$ be course j's preference to candidate i, both m and n have value between 0 and 1. The return value of score function should be:

$$\sum_i \sum_j \sigma_{i,j}(\lambda_1 m_{i,j} + \lambda_2 n_{j,i})$$

. Which λ_1 and λ_2 are different weights we consider course's and candidates's preferences to the other. Currently, we use 1 for both weights to value candidates and courses' opinions equally when scoring an assignment. All invalid assignments which breaks any of rule 1, 2, 3 at the end of section "Abstract" does not receive a score and should not be considered. (Theoretically, each algorithm described above should avoid generating an invalid assignment.)

5.2 Additional Metrics

In addition to sum the numerical preferences in the assignment, we also measure the percentage of courses and candidates which has their top-3 requests satisfied. In more detail, in an assignment, for a matched pair (a, b), which a is candidate and b is course:

- if a's preference score for b is within the top 3 scores of all a's preference scores, we consider b is a "top-3" choice for a. Same logic applied to b when checking if a is a "top-3" choices for a.
- If there are ties: suppose there are five courses: c1, c2, c3, c4, c5, and candidate a's preference score to these courses are: 0.9, 0.8, 0.8, 0.7, 0.6 in order, then the top 3 scores should be: 0.9, 0.8, 0.7. In this case c1, c2, c3, c4 all satisfy as "top-3" choices.

After getting the "top-3" satisfication count for both candidates and courses, we calculate the satisfiacation rate by:

$$\frac{\text{number of matches satisfying "top-3"}}{\text{all matches}} \times 100\%,$$

and "all matches" should be sum of capacity (number of TA wanted) of all courses.
We will use both the score, and preference satisfication rate of courses and candidates, to evaluate each method, and determine the best one.

6 Results

6.1 Simulations

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. Run the generated data on one (or all) of the three algorithms mentioned above.

Note: for the stable-marriages method, we coded the algorithm as described in an article from Cornell University; for the Hungarian Algorithm, we constructed the cost matrix as defined in section 4.2, and used the `scipy.optimize.linear_sum_assignment` package which exactly performs the Hungarian Algorithm; for the maximal matching algorithm, we used the `networkx.max_weight_matching` package.
For each set of random data (includes course preference, candidate preference, capacity and qualifications) generated, we will run all three algorithms on it, and record the evaluations (score, preference satisfication rate for both sides) for each method.

6.2 Simulation Result

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 Improvements

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.

Currently we value course's preference to candidates and candidates' preference to courses equally likely. We will test additional weights of them. This can be achieved by adjusting computation of the cost matrix in Hungarian Algorithm, and changed the direction of proposing in the Stable Marriage Algorithm, and change the value of λ_1 and λ_2 in the scoring function. We want to know the real-world implications if different weights are used.

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.

9 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.