Freshman Seminar Assignment Problem
Final Report

Keenan Gao **Binghui Ouyang** **Hanwen Zhang** **Yiming Zong**

Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

As the number of college students increases, an optimization algorithm that can automatically assign students to classes is becoming a pressing need in many universities. In attempts of solving the algorithm, various questions arised. What is the objective function to optimize? What are the aspects that need to be considered? In this final report, we will deliver solutions to the Freshman Seminar Assignment Problem, our team's final project for Operations Research II, Fall 2014. We will first briefly describe the real-world problem and create a basic mathematical model based on the data. Then, we will feed the pre-processed data into several different algorithms with various objective functions and run-time constraints, and compare their final results. Eventually, we will discuss some generalizations of our current problem and propose potential algorithms for solving them. Our algorithms improve the quality of assignment significantly in terms of student preference compared to the traditional manual approach.

1 Problem Overview

The assignment problem we are solving in this project was initially broached by Dietrich College of Humanities and Social Sciences at Carnegie Mellon University. The main goal is to assign freshmen to mendatory seminars in a way such that all seminars are filled, and that students are assigned to the seminars that they are interests in. In the real-world dataset that we are provided, there are 308 students and 22 seminars. Each seminar can have at most 16 students, and each student must enroll in exactly one seminar.

To facilitate the matching process, students are asked to rank one "first-choice" seminar and three "second-choice" seminars. We associate costs with assigning a student to a seminar based on the student's preference for the seminar. To find the optimal assignment, we approched the problem in two ways – minimizing the total cost across all seminars and smoothing out enrolled students' preference for each seminar.

2 Mathematical Model

Based on the problem description in *Section 1*, we can build a mathematical model for it in order to describe an alrogithm for solving the problem with any valid input data.

2.1 General Input

- n: Number of students ($n > 0$);
- m: Number of seminars ($m > 0$);
- k: Max number of selections that a student can make ($1 \leq k \leq m$);
- $s_{i,j}$: The j^{th} selection of i^{th} student, where $1 \leq i \leq n$ and $1 \leq j \leq k$. $s_{i,j} = 0$ when the Student i makes no corresponding choice for Rank j;
- q_k: The quota for k^{th} seminar, where $1 \leq k \leq m$.

2.2 Input Constraints

- Positivity: $n, m, k > 0, \forall k \in \{1, \cdots, m\}, q_k > 0$;
- Number of selections for student is bounded by number of available seminars: $k \leq m$;
- **(?)** Student rankings are valid and unique: $\forall (i, j), 1 \leq s_{i,j} \leq m$. And, for each i, all non-zero entries $s_{i,j}$'s take unique values.

2.3 Decision Variables

- $Y_{i,j}$: Indicator variables for whether Student i is assigned to Seminar j, where $1 \leq i \leq n$ and $1 \leq j \leq k$;

2.4 Data Pre-Processing

In order to deal with cases when a student is only willing or allowed to rank $k' < m$ seminars, we automatically set all "unassigned" priorities to $(k + 1)$. Also, we change the representation of students' preference from *(student, ranking)* $\mapsto$ *seminar* to *(student, seminar)* $\mapsto$ *ranking* to make further calculations easier, i.e.

$$X_{i,j} = \begin{cases} l & \text{If Student } i \text{ ranked } j \text{ as } l^{\text{th}} \text{ option, or } s_{i,l} = j \text{ for some } l \in \{1, \cdots, k\} \\ M & \text{If Seminar } k \text{ is not on Student } i\text{'s list, or } s_{i,l} \neq j \text{ for all } l \in \{1, \cdots, k\} \end{cases},$$

where M is an arbitrarily large value in order to discourage the algorithm from assigning a student to a seminar that s/he did not list.

2.5 General Constraints

- $Y_{i,j}$'s are indeed indicator variables: $\forall (i, j), Y_{i,j} \in \mathbb{Z}, Y_{i,j} \geq 0, Y_{i,j} \leq 1$;

- Each student is assigned precisely one seminar: $\forall i, \sum_{l=1}^{m} Y_{i,l} = 1$;
- Each seminar is within enrollment quota: $\forall j, \sum_{l=1}^{n} Y_{l,j} \leq q_j$;

3 Approach for Various Heuristic Functions

Due to the flexibility of the original problem, we are proposing different objective functions for optimization, including minimizing the total "rank" given by the students, maximizing the number of student getting their top λk choice (where $\lambda \in (0, 1)$), etc. In the following sub-sections we present our approach for each heuristic in mathematical terms.

3.1 Minimize Total Rank of Students

In this case, our goal is to minimize the sum of all student rankings for their assigned seminars. To do so, our objective is to minimize $W = \sum_{i=1}^{n} \sum_{j=1}^{m} X_{i,j} Y_{i,j}$.

3.2 Minimize Variance of Students Preference in Each Seminar

In addition to minimizing total rank of student, it would also be helpful if we could "balance out" students' preference of their assignd seminar for each seminar. For example, we do not want to have an assignment where some seminar has all students listing it as their first choice, yet some other seminar has none of the students listing it in their choices at all. With this idea in mind, for each seminar we wish to enforce a hard limit on the number of students enrolled that put it as first tier, second tier, etc.

Meanwhile, the hard limit can also affect the optimal solution that minimizes total rank of students in the final assignment. Therefore, care needs to be taken while picking the hard limits.

4 Exact Algorithm

The problem can also be reduced to an *Assignment Problem* when we include "dummy seminars" and "dummy students". When we apply *Hungarian Algorithm*, we can obtain an absolute optimal solution that minimizes the total "cost" of students. Details about the algorithm are as follows:

4.1 Data Pre-Processing

We start with the matrix $X_{i,j}$ as obtained in *Section 2.4*. For each column that represents Seminar j, we create extra $(j-1)$ dummy seminars by duplicating the same column q_j times. After that, we make our cost matrix square by adding zero rows at the bottom of the cost matrix. This gives us a matrix that we may feed into *Hungarian Algorithm*.

4.2 Hungarian Algorithm

Given the square cost matrix from previous section, we may simply apply *Hungarian Algorithm*, which returns a *student-seminar* assignment with minimal total cost. The algorithm completes in polynomial time `[reference]`, and given the result we may simply assign each student to the actual seminar that the assignment corresponds to.

5 Approximation Algorithms

While the previous exact algorithm minimizes the sum of ranks of students across all seminars, it does *not* balance out the preference of students in different seminars. In order to do so, a heuristic for approximation is to place an aritificial hard quota, $\mathcal{Q}$, on the number of students in any seminar that places it as its first option. While the optimal solution for different values of Q can be obtained by running Hungarian algorithm with different $\mathcal{Q}$'s by creating different combinations of dummy seminars, its runtime (more than one hour) makes it undesirable.

In this case, our solution is to use randomized greedy algorithm for arbitrarily number of times, and return the best solution to the user. Following is the detail of the algorithm:

5.1 Ranking-Based Greedy Algorithm

For the greedy algorithm, we first satisfy (a portion of) all students' first choices, then second choices, and so on. Depending on the "popularity" of each seminar, we may limit the number of students allowed to be added to a seminar at each ranking. The algorithm (as *Algorithm 1* on next page) is outlined as follows, and it can be run multiple times in order to select an assignment with least amount of students that are not assigned to their ranked list.

Algorithm 1 Ranking-Based Greedy Algorithm

Output: $\text{asgn}_i \leftarrow$ seminar assignment for Student i based on greedy algorithm
  for r = 1 to k **do**
    for i = 1 to m **do**
      pool[i] $\leftarrow \{$unassigned student s $\mid$ s listed seminar i as r^{th} choice$\}$
      pool[i] $\leftarrow$ random subset of itself with certain size limit (e.g. seminar quota)
    Merge each pool[i] into asgn
  Fill in still unassigned students

5.2 Stable Assignment Optimization

Similar to the principle of *Stable Marriage Problem*, in our final seminar assignment we do not want to have two students *A* and *B*, such that *A* prefers *B*'s section, and also vice versa (we call those two students *rogue pair*). This can be done by scanning each pair of students and fixing every *rogue pair*. The algorithm is outlined in *Algorithm 2* on the next page.

Algorithm 2 Rogue-Pair Fixing Algorithm

Input: $\text{asgn}_i \leftarrow$ current seminar assignment for Student i
Output: (i,j) if we found a rogue pair, otherwise null
  function FINDROGUEPAIR(asgn)
    for i=1 to n **do**
      for j=i+1 to n **do**
        if Student i and j prefer each other's seminar **then return** (i,j)
    return null

Input: $\text{asgn}_i \leftarrow$ seminar assignment for Student i based on greedy algorithm
Output: asgn_i: rogue pair-free assignment for Student i
  p $\leftarrow$ FindRoguePair(asgn)
  while p **not** null **do**
    (i,j) $\leftarrow$ p
    $\text{asgn}_i \leftrightarrow \text{asgn}_j$
    p $\leftarrow$ FindRoguePair(asgn)

6 Summary of Results

The algorithms are tested on the real data for the incoming class of Dietrich College for Year 2013 ($n = 309$). We have found that the approximation algorithm (...), while the *Hungarian Algorithm* gives really satisfactory assignment within five minutes. Despite the satisfactory performance of the exact algorithm, its run-time complexity $\mathcal{O}(n^3)$ makes the algorithm undesirable for $n > 1000$. Following is a comparison between the performance of manual assignment, approximation algorithm, and exact algorithm:

6.1 Approximation Algorithms

6.2 Naive Exact Algorithm

6.3 Smoothed Exact Algorithm

7 Further Work & Enhancements

7.1 Supporting bi-directional preference/cost parameters with Stable Marriage Algorithm

7.2 And more...

8 Acknowledgements

The authors would like to thank Professor Alan Frieze for proposing the Hungarian algorithm in *Section 5* and for holding weekly meetings to track our progress. Also, we thank Professor Brian W. Junker, Professor Joseph E. Devine, and Gloria P. Hill for providing us with the real seminar assignment data for Year 2013. And, the thank goes to Brian Clapper for his Python implementation of Hungarian algorithm, which made our final results and analyis possible.

References

[1] Martello, Silvano, and Paolo Toth. Knapsack Problems: Algorithms and Computer Implementations. Chichester: J. Wiley & Sons, 1990. Print.