Operations Research II
Final Report

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

Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Space reserved for abstract.

1 Problems Overview

`TODO: Overview of the problem in words.`

2 Mathematical Model

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 Maximize Number of Students Getting Top-Tier Seminars

In this case, we would additionally require the user to input a value $\lambda \in (0,1)$, representing the range of rankings we consider as "top-tier", i.e. $\{1, \cdots, \lfloor \lambda k \rfloor\}$. Given this heuristic, our objective is to maximize $W = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbb{1}_{X_{i,j} \leq \lfloor \lambda k \rfloor} Y_{i,j}$.

4 Approximation Algorithms

Given the constraints, our problem can be classified as a *Genrealized Assignment Problem*. According to Martello and Toth[1], it is *NP-hard*, so an approximation algorithm must be applied in order to solve the problem in a reasonable amount of time. Analogous to the *Knapsack Problem*, our fundamental approach is the greedy algorithm (with variations), and then make finishing touch based on the principle of *Stable Marriage Problem*. Following sub-sections will present the alrogithm in details:

4.1 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.

4.2 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 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 | 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

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)

---

5 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 rank of students. Details about the algorithm are as follows:

5.1 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*.

5.2 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.

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:

7 Further Work & Enhancements

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

7.2 And more...

8 Acknowledgements

Professor Frieze (weekly meeting, progress tracking)
Academic Advisor & Associate Dean from Dietrich (meeting about requirements, real data for Year 2013; names?)
Brian Clapper (Python implementation of Hungarian)
More?

References

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