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 [1], and given the result we may simply assign each student to the actual seminar that the assignment corresponds to.

5 Approximation Algorithm

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 $\mathcal{Q}$ 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 the balanced algorithm with user-defined number of iterations, and then return the best solution to the user. This algorithm allows us to determine a desired distribution of student interest and minimize its ensuing variance. Following is the detail of the algorithm:

5.1 Balanced Algorithm

For the balanced algorithm, we fix a value of $\mathcal{Q}$. For each seminar, we randomly select up to $\mathcal{Q}$ students who have selected it as their first choice, and assign them to that seminar. If the number of students who have ranked it first is smaller than $\mathcal{Q}$, then all of those students are assigned to that seminar. After each seminar has been filled with up to $\mathcal{Q}$ first choice students, we randomly fill the remaining $k - \mathcal{Q}$ seats in the seminar with the students who have ranked that seminar as their second choice. If there are students remaining after this second round of assignments, we then randomly assign them a seminar. The algorithm is outlined as follows, and it can be run multiple times in order to select an assignment that has the minimum total cost and variance of student interest.

Algorithm 1 Balanced Algorithm

Output: $\text{asgn}_i \leftarrow$ seminar assignment for Student i based on balanced algorithm
  while unassigned_students != 0 **do**
    for i = 1 to k **do**
      pool[i] $\leftarrow \{$unassigned student s $\mid$ s listed seminar i as r^{th} choice$\}$
    Merge each pool[i] into asgn
  Fill in still unassigned students

6 Summary of Results

So far we have tried two algorithms to solve the seminar assignment problem, namely the Exact Algorithm which applies the Hungarian Algorithm, and the Approximation Algorithm which utilizes randomization. Both algorithms are tested on the real data for the incoming class of Dietrich College for Year 2013 ($n = 308, m = 22$).The cost for assigning a student to a seminar is $g(x) = 2x^2$, where x is the tier that student ranks the seminar; if the student does not rank the seminar, the cost would be $g(x) = 10\,000$, which is a huge constant inspired by Big-M method.

6.1 Exact Algorithm

We use the Hungarian Algorithm to find the student-seminar assignment with the minimal total cost. The algorithm gives a very satisfactory assignment within five minutes. To be specific, among all the 308 students, 207 of them are assigned to their first choice of seminars, consisting of 67.2% of the population; and 82 are assigned to their second choice of classes, consisting of 26.6% of the population. Therefore, a total of 93.8% of the students are assigned to either their first or second choice of seminars. The remaining 19 students do not provide any preference of seminars, so they are randomly assigned to seminars which are not full. We also get a satisfying balance among seminar enrollments. Among all the 22 existing seminars, 18 of them get full enrollment of 16 students (16 was the highest enrollment allowed in 2013); one of the seminar get 15 students; one get 5 students; and the other two left seminars get 0 students enrolled. For the seminars that get 0 student, we would simply cancel it. The total cost of the optional solution is $1\,900\,164$.

6.2 Approximation Algorithms

The Approximation Algorithm returns a quite satisfying result with 10000 iterations under two minutes and a half. Among the 308 freshmen students, 151 of them are assigned to their first choices, which consists of 49.0% of the total population of students; 132 are assigned to their second choices, which consists of 42.9% of the population; 5 students are assigned to their later choices; and 19 of them do not show any preference of seminars and so are assigned to a random seminar which is still not full. Therefore, a total of 91.9% of the students who indicated preferences are assigned to

either their first or second choice. Regarding the balance of enrollments among the seminars, 15 of the seminars get full enrollment of 16 students; 3 of the seminars get pretty good enrollments of 15, 14 and 13 students respectively; and 3 of the seminars are poorly enrolled with 7, 5 and 2 students respectively. The total cost of the optional solution is $2\,200\,270$.

6.3 Comparison & Analysis

According to the results of the Exact Algorithm and the Approximation Algorithm, both are able to provide us with quite satisfying assignments. More than 90% of the students are able to get in their first or second choices of seminars. Also, the class enrollments are pretty balanced.

However, both algorithms have their own pros and cons. The Approximation Algorithm has a faster and scalable performance compared to the Exact Algorithm because it uses randomization instead of calculating the best result. Therefore, when the dataset gets really large, its better to choose the Approximation Algorithm. However, the relative performance of the Approximation Algorithm cannot be guaranteed compared to the Exact Algorithm, and it is absolutely not as good as the Exact Algorithm which always gives the minimum cost. Therefore, when the dataset is not huge and the time of using the Exact Algorithm is realistic, the Exact Algorithm should be a better choice as it provides us with the best solution.

6.4 Limitations

Despite the satisfactory performance of the exact algorithm, its run-time complexity $\mathcal{O}(n^3)$ makes the algorithm undesirable for $n > 1000$. Therefore, further work should be focused on more effective heuristics for the approximation algorithm in order to improve the quality of result under a reasonable time constraint.

7 Further Work & Enhancements

7.1 Allowing flexible input parameters

For our implementation, the main test case is for seminars with the same enrollment quota, and that each student makes one "first-choice" and three "second-choices". However, our implementation of algorithm in *Sections 4-5* also supports seminars with various quotas and also students that rank arbitrary number of seminars in arbitrary number of tiers. This makes our implementation applicable to much more real-world cases because in general the course sizes need not be the same, and that student should be given the opportunity to make flexible selections.

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

One of the features we could implement in the future might be that the final assignment does not only depends on the students' rankings on seminars but also the seminars' rankings on students. For example if a certain seminar is very major-orientated and prefers history major students mostly, then we will want to enhance our algorithm by accommodating for the bi-directional preferences. Algorithms like the Stable Marriage Algorithm would be useful to base on, or we may simply use the sum of cost in both students' and seminars' perspectives as parameters in the algorithms specified in *Sections 4-5*.

7.3 Stable Assignment Optimization

Similar to the principle of *Stable Marriage Problem*, in the 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 as follows:

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)

7.4 Parallelize Approximation Algorithm

Since the approximation algorithm in *Section Five* depends on a user-defined number of independent trials, we can actually parallelize the algorithm by using multiple threads, such that each worker is able to make trials and aggregate the result to the *Master* node. For a modern machine with multiple CPU cores, this is able to reduce the algorithm runtime by at least 50%.

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, and to Eric Wood, who implemented the excel-to-LaTeX utility.

References

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

Appendices

Appendix A: Raw Data And Results for Seminar Assignment Problem

Student ID:	**Student Selection:** Identify 1st choice and 3 2nd choices (ex., 7: 11, 15, 2)	**Assigned by Hand:** Notes: if blank none assigned	**Assigned by exact algorithm:**	**Assigned by approx algorithm:**
1	8: 14, 20, 3		3	3
2	14: 20, 12, 2		14	14
3	17: 13, 2		17	17
4	14: 6, 19, 13		13	13
5	12: 9, 20, 14		12	20
6	12: 20, 6, 3		12	12
7	14 : 2, 12, 11		14	14
8	12: 20, 6, 7		12	20
9	2: 10, 6, 3		3	3
10	2: 7, 6, 4		2	7
11	1	1	1	1
12	1	1	1	1
13	1	1	1	1
14	1: 15, 6, 13	1	1	1
15	1: 8	1	1	1
16	1	1	1	1
17		1	16	5
18	15: 8, 1, 9	1	15	15
19	1: 16, 15, 21	1	1	1
20	1: 17, 16, 10	1	1	16
21	1: 4, 18, 7	1	1	1
22	20: 18, 1, 12	1	20	20
23	1: 2, 9, 8	1	1	1
24	1	1	1	1
25		2	16	13
26	18: 14,13,9	2	18	18
27	2: 14	2	2	2
28	14: 7, 2, 12	2	14	2
29	14: 6, 2, 12	2	14	14
30	2: 10, 9, 3	2	3	3
31	2: 14, 4, 19	2	19	4
32	14: 2, 11, 10	2	14	11
33	7: 9, 14, 8	2	7	7
34	14: 2, 10, 6	2	14	10
35	2: 14, 15, 8	2	15	2
36	2: 14, 12, 16	2	16	16
37	8: 20, 3, 18	2	3	3
38	9: 14, 21, 4	2	9	9
39	2: 11, 7, 10, 14: 17, 12, 21	2	2	2
40	8: 9, 4, 20	3	8	8
41	2: 3, 6, 12.	3	3	3
42	2: 3, 10, 21	3	21	21
43	14: 2, 12, 13	3	13	13
44	18: 15, 20, 13	3	18	18
45	21: 14, 10, 6	3	21	21
46	9: 13, 5, 8, 12, 17, 21	3	9	17
47	6: 12, 18, 8	3	6	12
48	12: 13, 2, 18	3	12	13
49	2: 3,10,14,	3	3	3
50	8: 9, 10, 18	3	8	10

51	10: 8, 12, 14	3	10	12
52		4	16	1
53	4: 10, 2, 8	4	4	4
54	6: 12, 17, 9	4	17	17
55	1: 10, 3, 8	4	1	3
56	20: 15, 19, 16	4	20	20
57	10: 4, 2, 14	4	10	10
58	10: 4, 21, 9	4	10	10
59	6: 1, 14, 18	4	1	18
60	6: 2, 4, 8	4	6	6
61	4: 10, 21	4	4	4
62	19: 18, 9, 8	4	19	19
63	8: 4, 10	4	8	4
64	2: 14, 7, 6	4	2	14
65		6	16	3
66		6	16	5
67	6: 10, 21, 14	6	21	21
68	6: 5, 10, 7	6	6	6
69	6: 11, 9, 7	6	6	6
70		6	16	10
71	6: 8, 4, 10	6	6	4
72	6: 8, 7	6	6	7
73	20: 7, 6, 12	6	20	20
74	6: 20, 8	6	6	8
75	6: 9, 3, 7	6	3	7
76	6: 18, 9, 5	6	6	5
77	9: 20, 12, 6	6	9	9
78		7	17	19
79	14: 2, 20, 9	7	14	14
80	8: 6, 9	7	8	9
81	2: 12, 14, 8	7	2	2
82	2: 11, 6, 7	7	2	7
83	7: 2, 19, 14	7	7	7
84	15: 7, 18, 9	7	15	15
85	6: 7, 8, 3	7	3	6
86	8: 6, 9, 10, 20, 15, 12, 16	7	16	16
87	3: 6, 8, 5	7	3	3
88	12: 14 ,21, 2	7	12	12
89	8: 9, 6, 18	7	8	18
90	8: 6, 18, 7	8	7	7
91	1: 20, 6, 18	8	1	18
92	8: 9, 6, 12	8	8	12
93	8: 6, 20, 8	8	8	8
94	11	8	11	11
95	6: 8, 14, 2	8	6	8
96	8: 2, 14, 15	8	15	15
97	8: 7, 6, 10	8	7	8
98	8: 10, 11, 18	8	8	11
99	12: 9, 8, 14	8	12	12
100	18: 8, 15, 9	8	18	18
101	8: 10, 14, 4	8	8	4
102	8: 11, 6, 3	8	3	3
103	8: 7, 6, 9, 20	8	7	7
104		9	17	3
105	9: 18, 15, 16	9	9	16
106	8: 9, 20, 13	9	13	13

107	9: 6, 10, 8	9	9	8
108	9: 6, 8, 4	9	9	4
109	9: 8, 15, 10	9	9	9
110	9: 7, 14, 18	9	9	5
111	8: 18, 6, 9	9	8	9
112	9: 12, 20, 7	9	9	9
113	8: 9, 15, 18	9	15	8
114	11: 9, 2, 14	9	11	11
115	4: 9, 17, 18	9	4	4
116	9: 18, 16, 15	9	9	9
117	10: 4, 3, 7	10	10	7
118	12: 14, 21, 18	10	12	21
119	10: 15, 18, 3	10	10	10
120	8: 10, 6	10	8	6
121	10: 3, 1, 8	10	10	10
122	10: 21, 3, 1	10	10	21
123	19: 16, 9, 10	10	19	19
124	2: 10, 14, 20	10	2	20
125	10: 1, 21, 20	10	10	10
126	8: 18, 14, 16	10	16	8
127	10: 2, 11, 4	10	10	10
128	11: 2, 14, 20	11	11	14
129	11	11	11	11
130	11: 14, 20, 12	11	11	11
131	11: 10, 2, 17	11	17	17
132	11: 2, 14, 20	11	11	11
133	11	11	11	1
134	11: 2, 14, 12	11	11	12
135	11: 6, 1, 10, 19, 14, 15, 21	11	1	11
136	11: 14, 20, 9	11	11	20
137	2: 11, 14, 12	11	2	11
138	11: 2, 14, 18	11	18	3
139	11	11	11	11
140	11: 5, 2, 4.	11	11	5
141	2: 11, 10, 7, 5, 14, 21, 18, 20	11	21	7
142	11: 12, 14, 8	11	11	8
143	11: 2, 7, 18	11	7	11
144	20: 16, 6, 8	12	20	20
145	2: 14, 13, 12	12	13	13
146	12: 6, 17, 10	12	17	17
147	12: 9, 20, 6	12	12	12
148	1: 3, 6, 10, 12, 16, 19, 21	12	1	16
149	2: 11, 14, 15	12	15	15
150	20: 8, 6, 13	12	20	20
151	14: 10, 21, 4	12	21	14
152	13: 2, 12, 20	12	13	13
153	12: 21, 18, 14	12	12	21
154	6: 9, 8	12	6	6
155	12: 10, 2, 20	12	12	12
156	20: 14, 8, 12	12	20	20
157	11: 12, 2, 14	12	11	11
158	12: 10, 3, 2	12	12	2
159	11: 2, 14, 13	12	13	11
160	9: 8, 6, 11	13	9	9
161	14: 12, 16, 21	13	16	16
162	10: 11, 3, 20, 4	13	10	10

163	11: 12, 14, 2	13	11	14
164	8: 20, 14, 9	13	8	20
165	12: 13, 20, 2	13	13	13
166	13: 4, 8, 17	13	13	13
167	17: 12, 14, 15	13	17	17
168	6: 10, 14, 8	13	6	14
169	4: 18, 8, 10	13	4	4
170		13	17	19
171	7: 4, 2, 20, 12, 14	13	7	7
172	14: 20, 18, 12	13	14	12
173	2: 6, 7, 9	13	2	7
174		13	19	21
175		14	19	22
176	11: 13, 14, 20	14	13	13
177	9: 21, 17, 14	14	9	17
178	12: 17, 21, 19	14	19	12
179	13: 12, 2, 9	14	13	13
180	2: 4, 10, 14	14	2	4
181	2: 14, 7, 8	14	2	7
182	6: 4, 8, 15	14	6	15
183	2: 12, 14, 4	14	2	4
184	14: 9, 2, 19	14	19	14
185	14: 8, 20, 2	14	14	2
186	2: 14, 17, 18, 20	14	17	17
187	12: 14, 8, 6	14	12	14
188	14: 18, 6, 8	14	14	8
189	2: 14, 15, 10	14	15	2
190	9: 6, 12, 15	15	9	9
191	10: 8, 6, 3	15	10	3
192	15: 18, 9	15	15	15
193	15: 10, 18, 21	15	15	15
194	17: 14, 15, 21	15	17	17
195	4: 10, 2, 8	15	4	4
196	15: 7, 8, 2	15	15	15
197	15: 8, 9, 6	15	15	15
198	16: 15, 20	15	16	16
199	12: 13, 20, 7	15	13	12
200	21: 15, 6, 14	15	21	21
201	3: 16, 15, 3	15	3	3
202	18: 15, 9, 8	15	18	18
203	15: 18, 8, 9	15	15	15
204		16	19	13
205		16	19	16
206	8: 7, 14, 20	16	7	14
207	6: 14, 17	16	17	17
208	19: 13, 14, 16	16	19	19
209	8: 20, 18	16	8	20
210	8: 15, 16, 20	16	16	8
211	8: 3, 6, 14	16	3	8
212	10: 4, 23, 14	16	10	4
213	2: 7, 9, 4	17	2	2
214	17: 14, 21, 18	17	17	17
215	9: 11, 2, 13	17	13	13
216	18: 15, 7, 8	17	18	15
217	16: 19, 13, 12	17	16	16
218	6: 14, 12, 3	17	3	3

219	18: 12, 15, 17	17	18	17
220	9: 8, 6, 7	17	9	9
221	17: 9, 8, 18	17	17	17
222		17	19	21
223	11: 18, 12, 9	17	18	9
224	20: 8, 6, 15	17	20	20
225	6: 9, 8, 18	17	6	9
226	9: 6, 13, 17	17	17	17
227	16: 21, 15	17	16	16
228		18	19	22
229		18	19	1
230	12: 9, 18, 4	18	12	12
231	8: 9, 18, 7	18	7	8
232	11: 2, 8, 14	18	11	2
233	18: 8, 15, 6	18	18	18
234	18: 15, 14, 8	18	18	18
235	10: 12, 5	18	10	10
236	18: 21, 14, 8	18	18	18
237	12: 20, 14, 2	18	12	2
238	18: 14, 20, 2	18	18	18
239	8: 9, 18, 15	18	15	15
240	8: 12, 14, 2	18	8	8
241	2: 8, 14, 18	18	2	2
242	18: 15, 7, 8	18	18	18
243	13: 12, 20, 14	18	13	13
244	20:12	19	20	20
245	2: 10, 6	19	2	10
246	2: 12, 14, 16	19	16	16
247	14: 19, 21, 10	19	19	21
248	6: 10, 12, 16	19	16	6
249	2: 8, 21, 17	19	21	17
250	14: 19, 15, 20	19	19	14
251	6: 8, 9, 15, 12, 18	19	15	15
252	14; 11, 4, 2	19	14	4
253	8: 7, 9, 18	19	7	18
254	10: 6, 4, 18, 15, 20, 19	19	10	15
255	8: 14, 2, 7	19	7	8
256	19: 16, 12, 18	19	19	19
257	9: 20, 15, 6	19	9	9
258	3: 2, 20, 19	19	3	3
259	9: 2, 18, 13	20	13	13
260	6: 12, 20, 7	20	7	12
261		20	21	15
262	14: 10, 8, 18	20	14	18
263	20: 21, 18, 17	20	20	20
264	7: 20, 9, 17	20	7	7
265	20: 6, 14, 2	20	20	6
266	9: 2, 17	20	17	9
267	8: 7, 20, 6	20	8	6
268	20: 9, 8, 15	20	20	20
269	6: 8, 2, 10, 13, 12, 20, 18	20	13	18
270	14: 2, 8, 11	20	14	11
271	18: 14, 12, 9	20	18	18
272	14: 12, 20, 15	20	15	14
273	21: 17, 18, 10	21	21	21
274	8: 20, 21, 14	21	21	21

275	14: 2, 6, 4	21	14	14
276	12: 14, 13, 20	21	13	12
277	8; 6, 7, 20	21	20	6
278	12: 14, 15, 2	21	12	15
279	10: 3, 21	21	10	10
280	10; 11, 12 ,21	21	21	10
281		21	21	10
282	9: 20, 8, 6	21	9	6
283	21: 14,1,10	21	21	21
284	2: 6, 7, 21, 17, 14, 12	21	21	2
285	17: 13, 21,8	21	17	17
286	10: 21, 15	21	21	21
287	14: 3, 11	21	3	11
288		82-188/S14	21	16
289	14: 2, 8, 4	82-188/S14	14	4
290	7: 2, 8, 10	82-188/S14	7	7
291	10: 14, 8	82-188/S14	10	8
292	2: 7, 8, 4	82-188/S14	7	2
293	2: 4, 7, 9	82-188/S14	7	2
294	12: 20, 19	82-188/S14	20	19
295	6: 8, 14, 10	82-188/S14	6	6
296	20: 12, 13, 14	82-188/S14	20	13
297	2: 11, 12, 14	82-188/S14	2	11
298	14: 2, 8, 3	82-188/S14	14	14
299	12: 14, 2, 10	82-188/S14	12	12
300	9: 7, 20, 10	82-188/S14	20	7
301	11: 4, 2, 10	82-188/S14	11	4
302	12: 20, 9, 8	82-188/S14	20	9
303	6: 8, 10, 14	82-188/S14	6	6
304	6: 7, 8, 15	82-188/S14	15	6
305	6: 8, 9, 10	85-131/S14	6	6
306	9: 2, 6,18	85-131/S14	18	2
307	6: 8,9,18	85-131/S14	18	9
308	8: 6, 9, 10	85-131/S14	8	6

Appendix B: Source Code for Exact Algorithm with Hungarian

Dependencies:

- Python `numpy` package: available at http://www.numpy.org/;
- Python `Munkres` package: available at https://pypi.python.org/pypi/munkres/;

```
from munkres import Munkres, print_matrix
import numpy, sys
import re

def getCost(tier):
    return 2 * tier * tier

# Given a column number for augmented matrix, find its seminar number.
def getSeminar(index, q):
    s = 0
    for i in xrange(len(q)):
        s += q[i]
        if index < s:
            return i + 1
    return -1   # Shouldn't happen!

# Parse an input line to (seminar, tier) pairs
def parseLine(line):
    tierStrings = re.split(':|;', line)
    splitTier = [x.split(",") for x in tierStrings]
    splitTierResult = list()
    for tier in xrange(len(splitTier)):
        splitTierResult.append(list())
        for selection in splitTier[tier]:
            sanitizedSelection = re.sub("\D", "", selection)
            if re.sub("\D", "", sanitizedSelection) != "":
                splitTierResult[tier].append(int(sanitizedSelection))
    result = dict()
    for i in xrange(len(splitTierResult)):
        for s in splitTierResult[i]:
            if not(1 <= s and s <= m):
                print "Warning: Student entered out-of-bound Seminar ID: "
                    %s" % line
            elif s in result:
                print "Warning: Student has multiple entry for %d on line
                    %s" % (s, line)
            else:
                result[s] = getCost(i)

    return result

#
# !-- Entry point --!
#
# Specify input file name.
inputPath = "data/Fall2014"

# Ask for parameters from user
m = int(raw_input("Enter number of seminars (Seminar ID starts from 1): "
    ))
qq = raw_input("Enter quotas for %d seminars: " % m)
if len(qq.split()) == 1:
    q = [int(qq.split()[0]) for i in xrange(m)]
elif len(qq.split()) == m:
    q = [int(qq.split()[i]) for i in xrange(m)]
else:
    print "Invalid quota input!"
```

```
    sys.exit(1)

# Load student selections from input file.
print "Parsing input file '%s'..." % inputPath
with open(inputPath, 'r') as f:
    userInput = f.readlines()
if userInput[-1].startswith("END"):
    userInput = userInput[:-1]
else:
    raise Exception("Last line of file must be END!")
    sys.exit(1)
A = [parseLine(line) for line in userInput] # Parse input lines
n = len(A)
print "Number of students: %s" % n
if sum(q) < n:
    print "Quota cannot fit all students!"
    sys.exit(1)

# Transfer array to Student-Seminar.
B = []
MCOST = 100000
for i in xrange(n):
    B.append([A[i][j] if j in A[i] else MCOST for j in xrange(1, m+1)])
# Duplicate columns for hungarian
B = numpy.array(B, dtype='int32')
Slices = list()
for i in xrange(m):
    Slices.append(numpy.tile(numpy.transpose([B[:,i]]), q[i]))
C = numpy.concatenate(tuple(Slices), axis=1)
# Add zero rows for dummy students
for i in xrange(sum(q) - n):
    C = numpy.vstack([C, numpy.zeros(sum(q), dtype='int32')])
C = C.astype(int)

# Apply Munkres library to calculate Hungarian.
print "Running Hungarian algorithm on matrix of dimension", C.shape
C = C.tolist()
m = Munkres()
indexes = m.compute(C)
total = 0
print "Student ID, Assigned Seminar, Cost"
for row, column in indexes:
    if row >= n: continue # Skip dummy students
    value = C[row][column]
    total += value
    print '%d, %d, %d' % (row, getSeminar(column, q), value)
print 'Total Cost: %d' % total
```

Appendix C: Source Code for Randomized Approximation Algorithm

Dependencies:

- Python `numpy` package: available at http://www.numpy.org/;

```
import math, numpy, sys
import re
import copy
import random

def getCost(tier):
    return 2 * tier * tier

# Parse an input line to (seminar, tier) pairs
def parseLine(line):
    tierStrings = re.split(':|;', line)
    splitTier = [x.split(",") for x in tierStrings]
    splitTierResult = list()
    for tier in xrange(len(splitTier)):
        splitTierResult.append(list())
        for selection in splitTier[tier]:
            sanitizedSelection = re.sub("\D", "", selection)
            if re.sub("\D", "", sanitizedSelection) != "":
                splitTierResult[tier].append(int(sanitizedSelection))

    result = dict()
    for i in xrange(len(splitTierResult)):
        for s in splitTierResult[i]:
            if not(1 <= s and s <= m):
                print "Warning: Student entered out-of-bound Seminar ID: 
                    %s" % line
            elif s in result:
                print "Warning: Student has multiple entry for %d on line
                    %s" % (s, line)
            else:
                result[s] = getCost(i)

    return result

# Main function for running randomized assignment.
def randomAsgn(quotaRatio):
    roster = [list() for i in xrange(m+1)]
    currCost = 0
    currAsgn = dict()
    availStudents = set(range(n))
    # Stage One: Assign first choices, up to set quota.
    for s in xrange(1, m+1):
        tmp = randomSub(set(R[s][0]) & availStudents,
                int(math.floor(float(q[s-1]) * quotaRatio)))
        roster[s] += tmp
        currCost += getCost(0) * len(tmp)
        for student in tmp:
            currAsgn[student] = s
        availStudents = availStudents.difference(set(tmp))
    # Stage Two: Assign second choices, as many as possible.
    for s in randomly(xrange(1, m+1)):
        tmp = randomSub(set(R[s][1]) & availStudents, q[s-1] - len(roster
            [s]))
        roster[s] += tmp
        currCost += getCost(1) * len(tmp)
        for student in tmp:
            currAsgn[student] = s
        availStudents = availStudents.difference(set(tmp))
    # Stage Three: Enroll remanining students in Round-robin fashion.
```

```
    s = 1
    while len(availStudents) > 0:
        if len(roster[s]) < q[s-1]:
            student = availStudents.pop()
            roster[s].append(student)
            currCost += MCOST
            currAsgn[student] = s
        s = 1 if s == m else s+1
    return (currCost, currAsgn, roster)

# Getting random sublist of a set with length l
def randomSub(L, l):
    return random.sample(list(L), min(len(list(L)), l))

# Gives random iterator of a list
def randomly(seq):
    shuffled = list(seq)
    random.shuffle(shuffled)
    return iter(shuffled)


#
# !-- Entry point --!
#
# Specify input file name.
inputPath = "data/Fall2014"

# Ask for parameters from user
m = int(raw_input("Enter number of seminars (Seminar ID starts from 1): "
    ))
qq = raw_input("Enter quotas for %d seminars: " % m)
if len(qq.split()) == 1:
    q = [int(qq.split()[0]) for i in xrange(m)]
elif len(qq.split()) == m:
    q = [int(qq.split()[i]) for i in xrange(m)]
else:
    print "Invalid quota input!"
    sys.exit(1)
iters = int(raw_input("Enter number of iterations to run randomized
    assignment: "))

# Load student selections from input file.
print "Parsing input file '%s'..." % inputPath
with open(inputPath, 'r') as f:
    userInput = f.readlines()
if userInput[-1].startswith("END"):
    userInput = userInput[:-1]
else:
    raise Exception("Last line of file must be END!")
    sys.exit(1)
A = [parseLine(line) for line in userInput] # Parse input lines
n = len(A)
print "Number of students: %s" % n
if sum(q) < n:
    print "Quota cannot fit all students!"
    sys.exit(1)

# Transfer input to (student,seminar) -> ranking mapping.
# Initialize (seminar, ranking) -> student mapping.
B = []
R = [([], [])]
MCOST = 100000
for i in xrange(n):
    B.append([A[i][j] if j in A[i] else MCOST for j in xrange(1, m+1)])
for i in xrange(1, m+1):
```

```
        rankedFirst = list()
        rankedSecond = list()
        for s in xrange(n):
            if i in A[s] and A[s][i] == getCost(0):
                rankedFirst.append(s)
            elif i in A[s] and A[s][i] == getCost(1):
                rankedSecond.append(s)
        R.append((rankedFirst, rankedSecond))

    # Mainloop for iterations
    bestCost = sys.maxint
    bestAsgn = list()
    roster = list()
    bestRatio = 0.0
    for i in xrange(iters):
        for j in xrange(1, max(q)+1):
            (currCost, currAsgn, currRoster) = randomAsgn(j * 1.0 / max(q))
            if currCost < bestCost:
                print "Current optimal cost: %d ... [Run %d out of %d]" % (
                    currCost, i+1, iters)
                bestCost = currCost
                bestAsgn = copy.deepcopy(currAsgn)
                bestRoster = copy.deepcopy(currRoster)
                bestRatio = j * 1.0 / max(q)

    print "Best Cost: %d  (with first-choice quota %f)" % (bestCost,
        bestRatio)
    print "Assignment as follows:"
    for i in xrange(n):
        print "%d -> %d" % (i, bestAsgn[i])
    print "-----------------------------------"
    print "Seminar roster:"
    for i in xrange(1, m+1):
        print "Seminar %d:" % i, bestRoster[i]
```