Carnegie Mellon University

21-393 Operations Research II Final Project

Optimizing Dietrich College Freshman Seminar Assignments

Authors:
Paul KIM, Shannon JUNG, Noelle JUNG

Advisor:
Dr. Alan FRIEZE

Mellon College of Science
Department of Mathematical Sciences

December 2014

Abstract

This paper investigates the problem of optimizing Carnegie Mellon University's Dietrich College freshman seminar assignments. As Dietrich College faces this task every semester and has hoped to improve the current process, we seek to explore the problem and devise a more efficient method that will successfully optimize freshman seminar assignments. In doing so, we have applied algorithms learned from the Operations Research II class to formulate two objective functions that allowed us to approach the problem from different aspects and compare the results in hopes to produce the most optimal solution.

Acknowledgments

We would like to show our sincere appreciation to Dr. Brian Junker, Dr. Joseph Devine, and Dr. Gloria Hill for providing real survey response data, explaining the current situation and its problems, and meeting with us to answer all of our questions; Dr. Frieze for guiding us throughout the project; and Brian Clapper for constructing and uploading Munkres.py, an external Python library that implements the Hungarian algorithm.

Contents

Abstract 2

Acknowledgments 3

1. Problem Statement 5
   1.1 Introduction 5
   1.2 Model 5
2. Data 7
   2.1 Data Simulation 7
   2.2 Real Data 8
   2.3 Current Issues and Suggestions 9
3. Implementation 11
   3.1 Hungarian Algorithm 11
   3.2 Constructing the Cost Matrix 12
   3.3 Solving the Objective Function 14
4. Results 15
   4.1 Simulated Data 15
   4.2 Real Data 16
5. Conclusion 17

A Appendix - Survey Response Format 18

B Appendix - Code 19

1. Problem Statement

1.1 Introduction

All freshmen in Dietrich College are required to take one of about 22 seminars in their first year. Each of the 350 or so freshmen in Dietrich College submit their preferences before registering online. These data are currently being left unused, resulting in several avoidable inefficiencies. For example, waitlists may grow while some seminars remain unfilled, or the number of students registering for a seminar in the fall may greatly outweigh that in the spring. In both cases, staff may need to add extra seminars, creating new problems of finding available rooms, time slots, and instructors.

Our team approached the Dietrich College staff to research ways to efficiently reach a solution that maximizes student satisfaction while meeting, among others, the following criteria. The staff desired an approximate assignment, which advisors could tweak and use to encourage students to register for specific seminars. It was important that all students be assigned to one of their top choices and that all seminars be filled evenly with 15-16 students each. In addition, the staff desired a balance between seminars being taught in the fall and spring.

Our team took on this task by modeling the problem as an assignment problem, simulating student preference data to fit our model format, and applying the Hungarian algorithm to reach the optimal solution for two different objective functions. Along the way we also input real preference data provided by Dietrich College and explored one option for data collection.

1.2 Model

Let I be the set of Dietrich College freshmen and let J be the set of seminars. Let c_{ij} be student i's ranking of seminar j. Our model permits each student to rank his top 5 seminars, with $c_{ij} = 1$ denoting the student's first choice and $c_{ij} = 5$ denoting the fifth choice. Rankings for the remaining classes are automatically set to $c_{ij} = 100$. Our model

defines two possible objective functions: minimizing the total of the assigned seminar rankings or minimizing the maximum assigned seminar ranking.

Decision variables:

$x_{ij} \in \{0, 1\}$	A binary variable indicating if student $i \in I$ is assigned to seminar $j \in J$

Constraints:

$\sum_{j \in J} x_{ij} = 1 \ \forall \ i \in I$	Each student is assigned to exactly one seminar.
$15 \leq \sum_{i \in I} x_{ij} \leq 16 \ \forall \ j \ \in J$	Each seminar has 15-16 students assigned to it.

Objective Functions:

1) Minimize $\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$	Minimize the total of the rankings for each student's seminar assignment.
2) Minimize $max_{i \in I, \, j \in J} \{c_{ij} x_{ij}\}$	Minimize the maximum ranking of all students' seminar assignments.

2. Data

2.1 Data Simulation

After our first meeting with Dr. Junker, we were informed that the Dietrich College staff would need a few weeks to gather the most recent survey response data. In order to proceed with our project, we decided to generate a data set on our own, using the facts that there are approximately 350 freshmen and that around 11 seminars are offered in each semester. In our simulation, we set each student to pick two seminars from each semester and the last seminar randomly, then shuffle the result to randomize the order which represents each student's seminar preference ranking. We implemented this algorithm in Python to generate our data set. In the output, one student is a list of five strings that represents seminars in order of preference. The output was, as we expected, evenly spread out as shown in the bar chart (Figure 1) below.

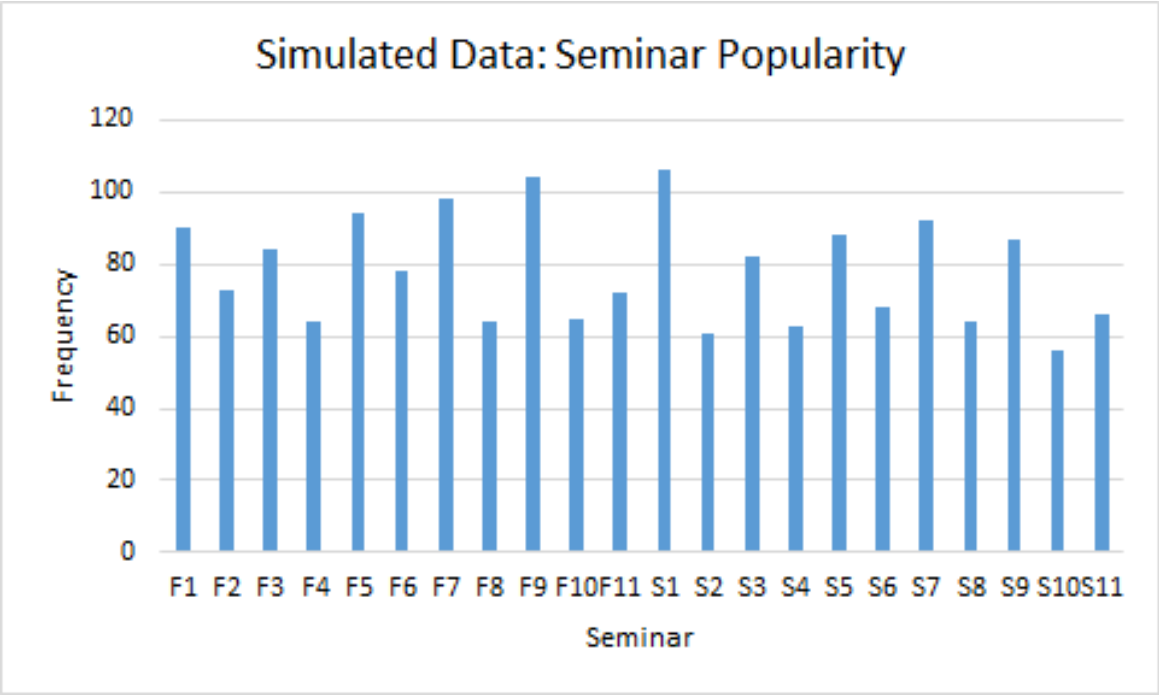


Figure 1

Since our simulation was designed such that all students choose exactly five seminars, we expected that the number of times that a certain seminar is chosen would be higher than in a real data set. Also we were aware that our data did not reflect that in reality, some seminars are more/less popular than the others.

2.2 Real Data

After our second meeting with the Dietrich College staff, we were provided with a real data set consisting of every freshman's seminar preferences. The real data set differed from the simulated data set in that each student named one seminar of first choice and three seminars of second choice (ranked 4 seminars in total).

Because the students' responses on SurveyMonkey were manually transferred to a word document, we noticed a few human errors that occurred in the real data set we received. The table below in Figure 2 is a section from the real data set containing some erroneous data (highlighted in yellow). The cells indicated as no data and incomplete data were resulted for some students failed to rank or fully rank the seminars, whereas the cells indicated as misplaced data or typo were resulted due to the manual transferring of data.

	Student ID	Preferences
	...	...
	24	1
No data	25	
	26	18: 14,13,9
Incomplete data	27	2: 14
	28	14: 7, 2, 12
	29	14: 6, 2, 12
	30	2: 10, 9, 3
	31	2: 14, 4, 19
	32	14: 2, 11, 10
	33	7: 9, 14, 8
	34	14: 2, 10, 6
	35	2: 14, 15, 8
	36	2: 14, 12, 16
	37	8: 20, 3, 18
	38	9: 14, 21, 4
Misplaced data	39	2: 11, 7, 10, 14: 17, 12, 21
	40	8: 9, 4, 20
Typo	41	2: 3, 6, 12.
	...	...

Figure 2

Using the real data set, we then generated a bar graph (Figure 3) that represents the popularity of each seminar - this was done by counting the number of times a certain seminar was mentioned in the real data set. Observe that the total number of frequencies of seminars is less than that of the simulated data. This is because the students ranked 4 seminars in the real data whereas 5 seminars were ranked in the simulated data. Also, we did not take into account the students that failed to respond or complete the rankings.

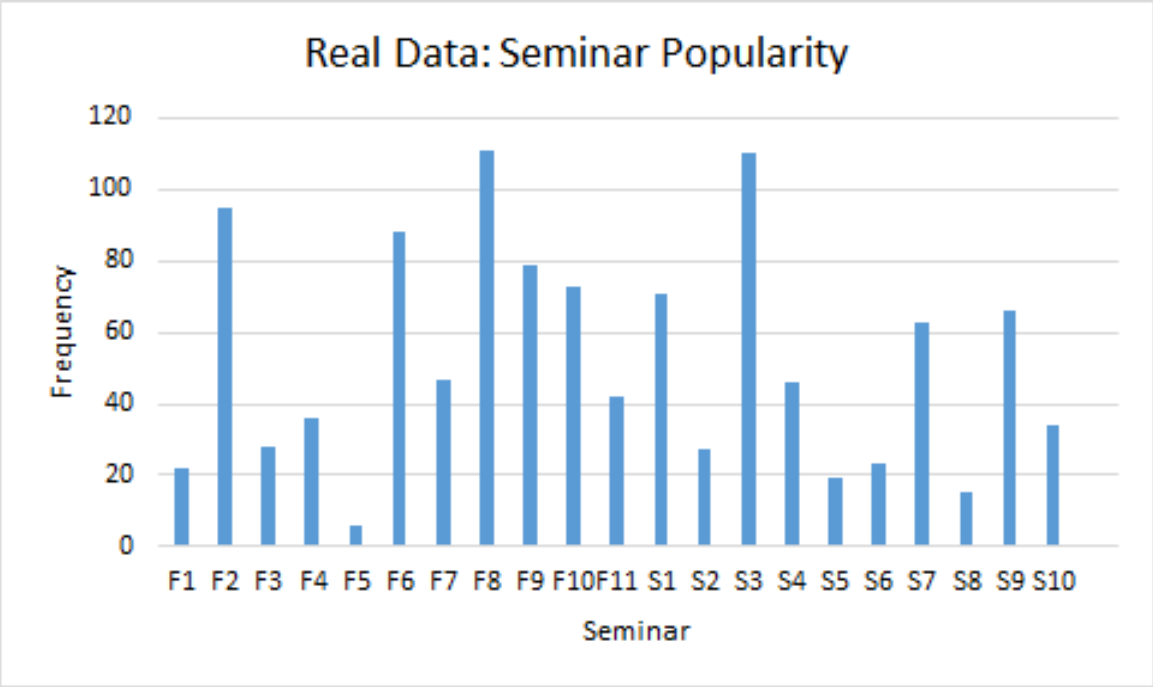
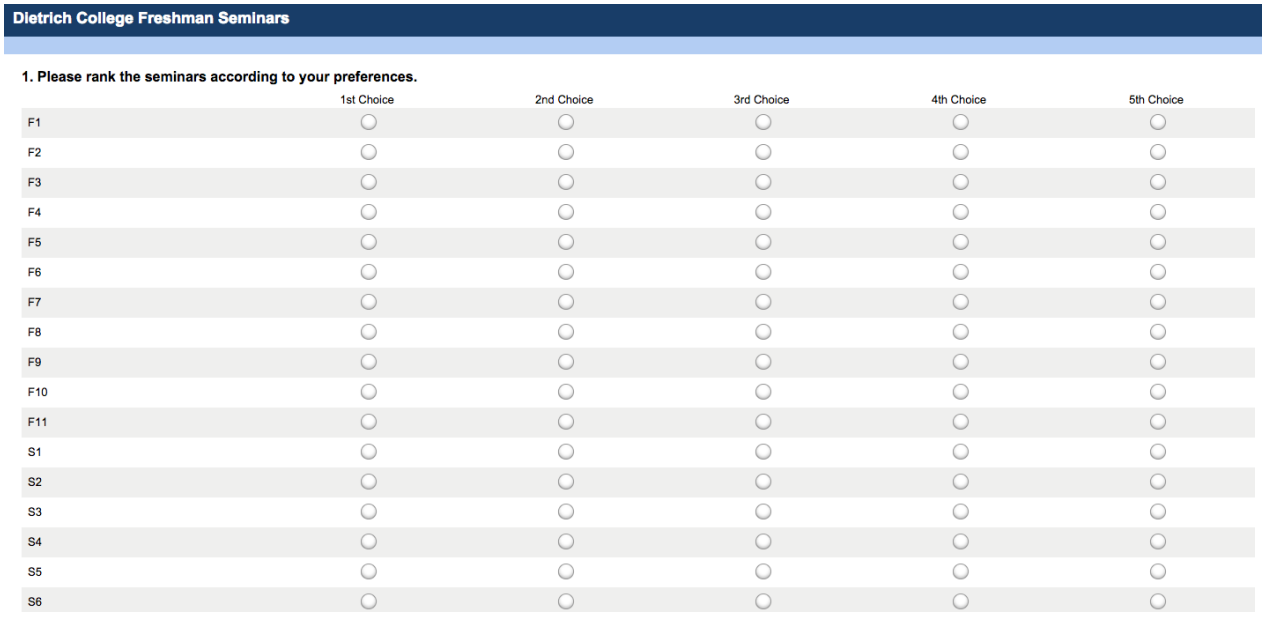


Figure 3

2.3 Current Issues and Suggestions

As mentioned in the previous section, the current method of collecting data was inefficient and inaccurate. We needed a "better" way. With Dr. Junker's permission, we decided to create our own Survey Monkey page like the page shown in Figure 4. Note that this page is what a student sees to participate in the survey. One student is allowed to check one seminar per one column.

As the survey page shows, each student is to rank his/her top five choices. The reason we kept this way of ranking is that by the time we saw the real data format, it was too late to fix all of previously written codes. Also Dr. Junker strongly recommended to make our own format if the change could benefit result in any way. And we believe this way of ranking would help us analyzing student satisfaction more thoroughly.



Dietrich College Freshman Seminars

1. Please rank the seminars according to your preferences.

	1st Choice	2nd Choice	3rd Choice	4th Choice	5th Choice
F1	○	○	○	○	○
F2	○	○	○	○	○
F3	○	○	○	○	○
F4	○	○	○	○	○
F5	○	○	○	○	○
F6	○	○	○	○	○
F7	○	○	○	○	○
F8	○	○	○	○	○
F9	○	○	○	○	○
F10	○	○	○	○	○
F11	○	○	○	○	○
S1	○	○	○	○	○
S2	○	○	○	○	○
S3	○	○	○	○	○
S4	○	○	○	○	○
S5	○	○	○	○	○
S6	○	○	○	○	○

Figure 4

After reading Survey Money manual page, we noticed that data can be exported automatically as an Excel file which would reduce the time of process and eliminate human errors. Moreover, since the outputs are in a standardized format explained in the manual, we could anticipate the format of the results (more details in Appendix A).

However, usage of the survey page had limits. Premium membership, a paid service which the Dietrich College staff did not have, was required to export data in any file type.

3. Implementation

Our group's primary focus was to create a practical solution in a format familiar to the Dietrich College advisors that could be easily used to produce a readable assignment scheme. We first explored the option of writing an Excel plugin that would harness the features of the Excel Solver add-in, which provides several linear programming engines and would have been extremely convenient to use. However, our problem size exceeded Solver's limits, so we considered coding an assignment algorithm in VBA.

We decided to produce a solution written in Python before diving into a VBA plugin, as we were more familiar with Python and had no prior experience with VBA. To achieve our usability goals, we focused on a Python script that would take an Excel worksheet with preference data, find solutions to two objective functions, and write the resulting assignments to a new Excel workbook.

Our implementation was comprised of 2 main components:

1. A Python program to create a cost matrix from the student response data, run the Hungarian algorithm, and output the results in a new Excel workbook (included in the appendix as solve.py)
2. An external library called munkres.py written by Brian Clapper that applies the Hungarian algorithm to an input cost matrix

Figure 5 shows how these components work together.

For our program to read the student response data we assumed that the input Excel worksheet would follow the format defined by Survey Monkey, the current survey method of choice. This format is outlined in Appendix A.

3.1 Hungarian Algorithm

The Hungarian Algorithm is an algorithm that solves the assignment problem in polynomial time, and is the central algorithm used to solve the main problem in our project. The algorithm requires an n x n cost matrix, where the entry at i^{th} row and j^{th} column is the cost of assigning the i^{th} element (from the rows) to the j^{th} element (from

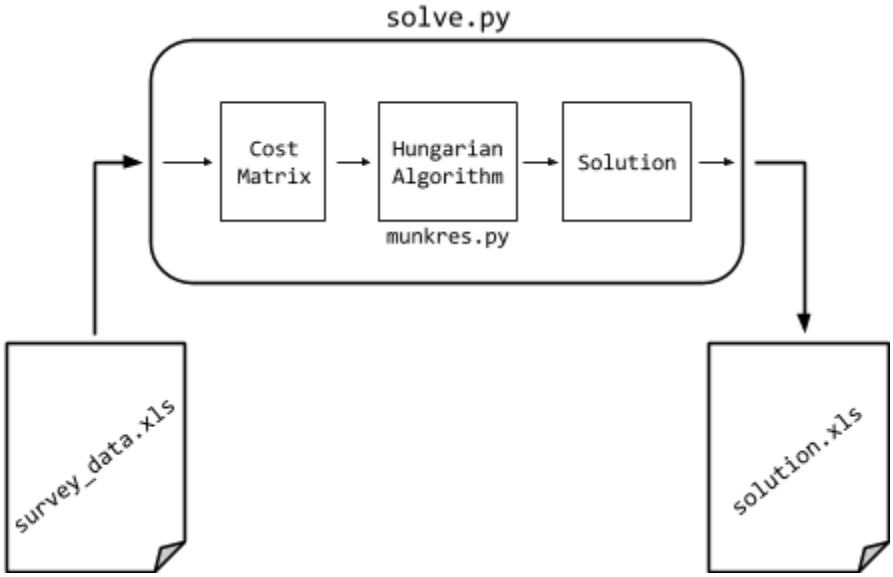


Figure 5

the columns). The algorithm then produces a one-to-one assignment with minimum total cost. Graphically, a bipartite graph of equal number (n) of vertices will be required, and the Hungarian Algorithm will yield a complete one-to-one matching with the total weight of the edges minimized.

In the process of trying to implement the Hungarian Algorithm, we came across the Munkres module, which is a Python library that provides an implementation of the Hungarian Algorithm. It can be seen in the attached code in Appendix B that the Hungarian Algorithm was implemented with the Munkres library.

3.2 Constructing the Cost Matrix

As explained above, we needed an n x n cost to matrix to run the Hungarian Algorithm. So we constructed it in the following way.

First, we filled up all the empty entries (seminars not in the top five choices of a particular student) of the survey response matrix with a heavy cost, in our case 100, to get a cost matrix like in Figure 6.

Second, we decided to duplicate the entries "number of students / number of seminars" times, rounding down to an integer. If we still needed more columns to produce an n x n matrix, we filled the columns with the most popular seminars in order of their popularity. This case was when "number of students / number of seminars" was not an

integer (i.e. students did not divide evenly across the seminars, creating the need for extra seats in some seminars), and the number of extra columns we put was less than

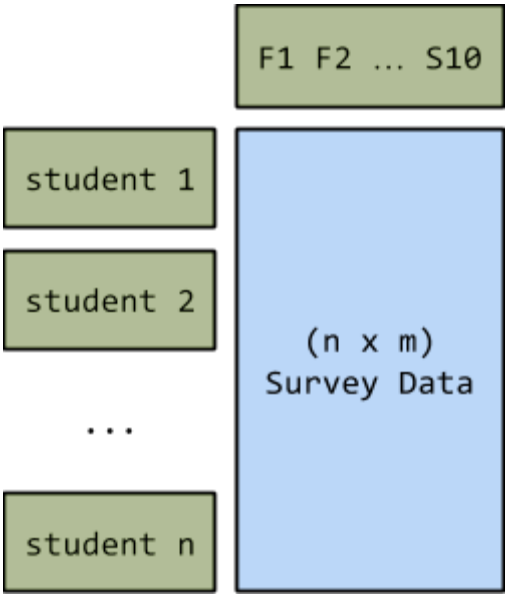


Figure 6

the number of seminars. Then we would have an n x n matrix in which each column represented an available seat in a seminar. In this way, if 15-16 students were to be assigned for each seminar, popular seminars would have 16 students (the upper bound). The popularity of a seminar was defined by the number of times it appeared on all students' top five choices. The popular seminars are indicated by * in Figure 7 below.

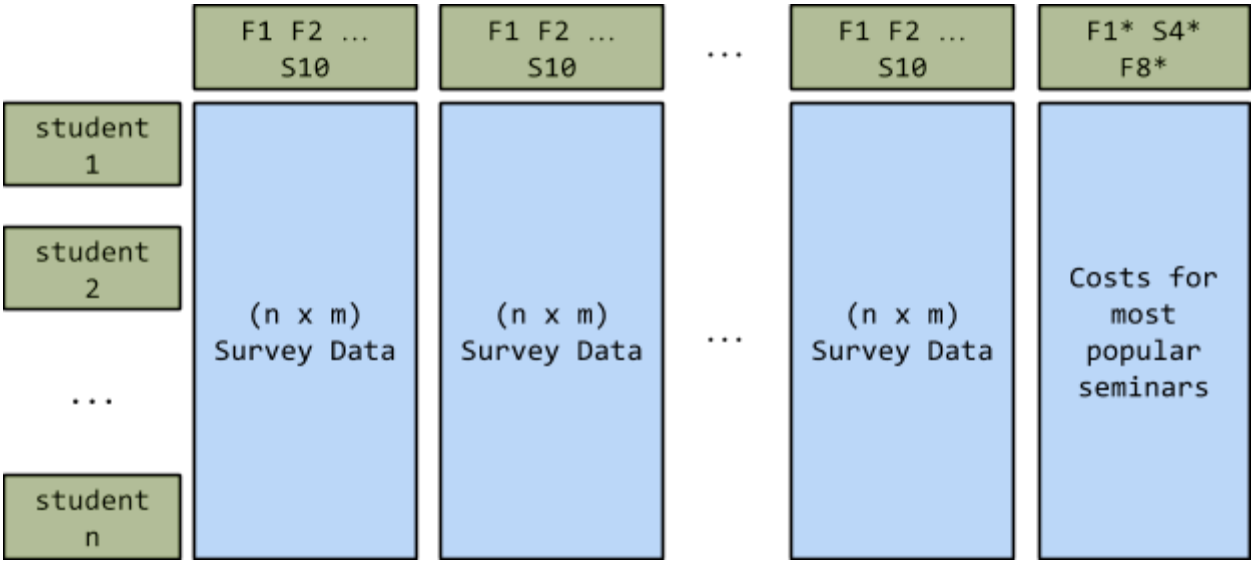


Figure 7

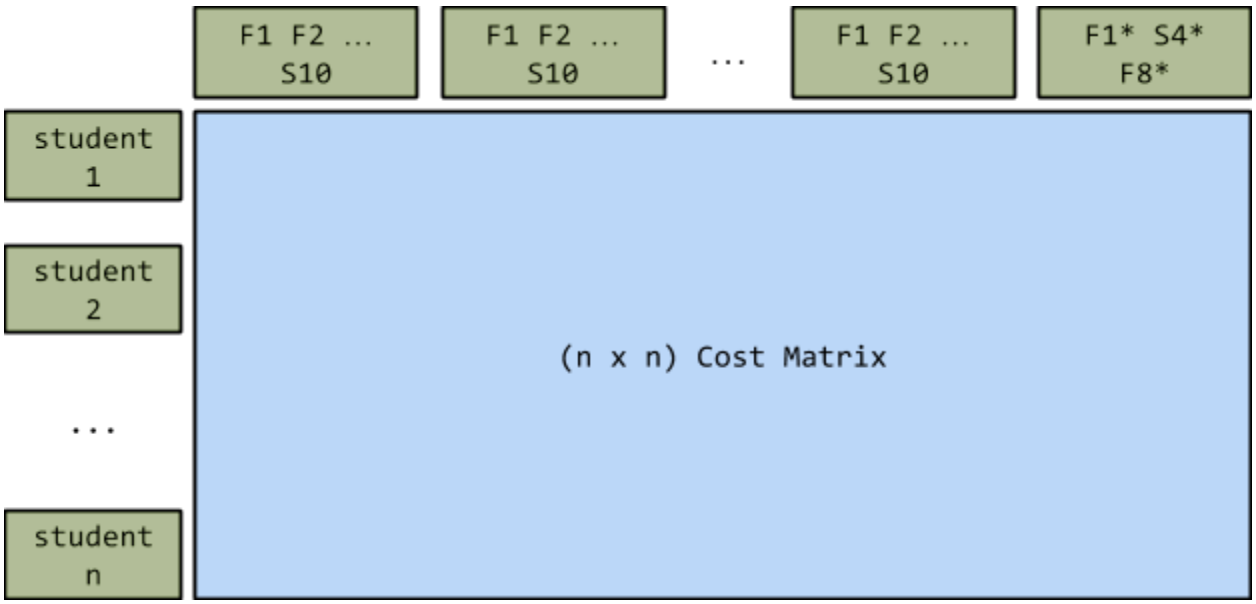


Figure 8

We were able to produce the n x n cost matrix as in Figure 8 above, and apply the Hungarian Algorithm.

3.3 Solving the Objective Function

Our implementation solves two separate objective functions. The first minimizes total cost and the second minimizes the maximum cost. Solving the first objective function simply requires running the Hungarian algorithm once. For the second objective function, we repeatedly run the Hungarian algorithm while incrementing a level, k, that defines the maximum cost allowable (e.g. for $k = 3$, all seminar assignments must be the student's third choice or better). At each step we alter the cost matrix so that any costs greater than k are set to 100, run the Hungarian algorithm, and check if the solution includes any 100-cost assignments. If so, the solution is invalid so we increment the level and repeat the process. If not, the solution guarantees students a seminar that is their k^{th} choice or better, and we end the loop. This process runs a maximum of five times, as students are allowed to rank their top five seminars.

4. Results

4.1 Simulated Data

Minimum Total Cost

Student	Seminar	Original Ranking
1	F02	1
2	S09	1
3	F06	2
4	S01	1
5	S04	1
6	S09	1
7	S08	1
8	F07	1
9	F04	1
10	S04	1
11	F07	1
	…	
115	S01	4

Total Cost: 366
Max Cost: 4

Minimum Max Cost

Student	Seminar	Original Ranking
1	F02	1
2	S09	1
3	S03	1
4	S01	1
5	S04	1
6	S09	1
7	S08	1
8	F07	1
9	F04	1
10	S04	1
11	F07	1
	…	
115	S07	2

Total Cost: 366
Max Cost: 4

The total costs from the two different objective functions were extremely similar. The minimum total cost was 366 and the minimum max total cost was 367. As the two tables above indicate, it was obvious just by looking at the two results that the majority of students were assigned their first choices. Moreover, computationally, 350 students having a total cost of 366 and 367 means that, except a few, every student got into his or her favorite seminar. Because the simulated data was unrealistically balanced, we believe the result also was unrealistically optimized.

The maximum cost of the min total cost objective was 4, while the maximum cost of the min max cost objective was 2. It was a trivial result considering how algorithms that

minimize total cost and max cost run according to the two different objective functions.

4.2 Real Data

Minimum Total Cost

Student	Seminar	Original Ranking
1	F08	1
2	S09	2
3	S06	1
4	S08	3
5	S01	1
6	S01	1
7	S03	1
8	S01	1
9	F10	2
10	F07	2
11	F01	1
12	F01	1
13	F01	1

Total Cost: 2340
Max Cost: 100

Minimum Max Cost

Student	Seminar	Original Ranking
1	F08	1
2	S09	2
3	S06	1
4	S08	3
5	S01	1
6	S01	1
7	S03	1
8	S01	1
9	F10	2
10	F07	2
11	F01	1
12	F01	1
13	F01	1

Total Cost: 2340
Max Cost: 100

It can be seen from the two tables above that the the results obtained by the two objective functions from the real data are identical with the total cost of 2340 and maximum cost of 100.

In contrast to the simulated data, it can be observed that the results from the real data do not guarantee all students' top 4 choices. The maximum cost of 100 illustrates this fact, and is due to the real data's unevenly distributed popularities of the seminars (a realistic distribution compared to that of the simulated data).

5. Conclusion

Initially, we were presented with a freshmen seminar assignment process that has been done manually without considering students' preferences for seminars. Through various meetings with the Dietrich College staff and Dr. Frieze, we were able to construct a process that produces an optimal assignment with increased efficiency and accuracy. We also incorporated

Although our solution successfully returns a solution, it makes three key assumptions and faces some limitations as a result. First, our program must be run from a terminal window on a system that has Python and a few external libraries already installed. Second, we assume that survey results are collected through a Survey Monkey survey that follows our specific question format. In addition, paid membership to Survey Monkey is required for extracting Survey Monkey results in an Excel file. This leads us to our third assumption. Our program assumes that the survey response data adheres to the Survey Monkey format. If this assumption is not met the assignment solver cannot extract the data and run the Hungarian algorithm.

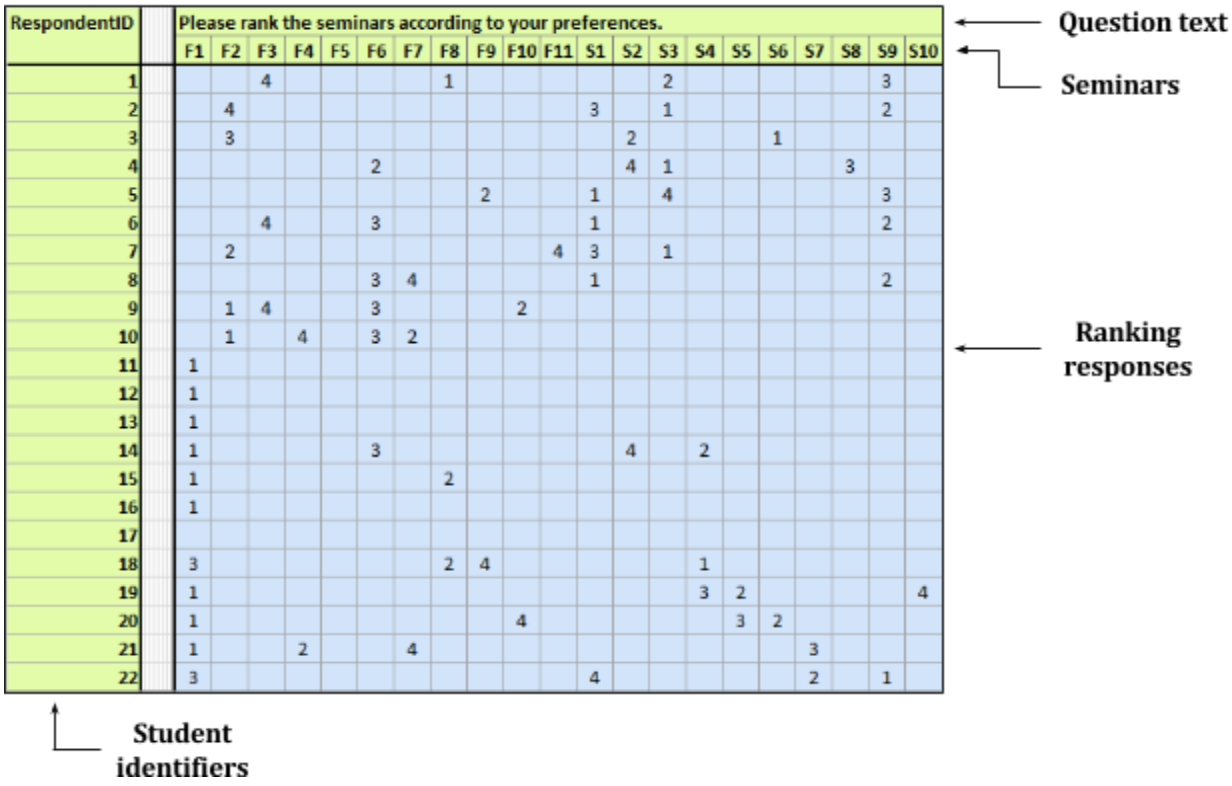
For further steps, our project would be more practical if we produce an Excel plugin so that the Dietrich College staff can run our program without accessing through terminal.

Second, we need to construct an actual survey page that collects student data in the format that we wants and export the result in Excel file automatically. We also thought constructing a funny survey page that captures participant's attention would reduce the number of student did not respond or complete the survey.

Third, generating differently skewed data, such as data with extremely popular seminars or semester, would be helpful. In this way, we would be able to test our two objective functions under various conditions and study the difference more thoroughly so that we can indicate which function to use in certain situation.

Appendix A - Survey Response Format

Survey responses that are fed into our assignment solver must follow the format defined by the "All Responses Data" option from Survey Monkey. Figure 9 below summarizes this format. Full details can be found at http://help.surveymonkey.com/articles/en_US/kb/XLS-Exports.



RespondentID	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
1			4					1						2						3	
2		4										3		1						2	
3		3											2				1				
4						2							4	1					3		
5									2			1		4						3	
6			4			3						1								2	
7		2									4	3		1							
8						3	4					1								2	
9		1	4			3				2											
10		1		4		3	2														
11	1																				
12	1																				
13	1																				
14	1					3							4		2						
15	1							2													
16	1																				
17																					
18	3							2	4						1						
19	1														3	2					4
20	1									4						3	2				
21	1			2			4											3			
22	3											4						2		1	

Figure 9

Note that each row represents one student's preference ranking. Since this table reflects the real data, the ranking only went up to 4.

Appendix B - Code

Code for the assignment solver program can be found at https://github.com/njung92/assignmentSolver.