

Optimal Assignment of Medical Students to Branch Campuses

STOR 765
Spring 2023
March 29, 2023

Hank Flury
Peter Lin

Client: UNC School of Medicine (Valerie Glassman, Mary Hauser)

Abstract

In previous years the assignment of medical students to branch campuses has been done through a random lottery. The goal of this work is to optimize this assignment and improve student satisfaction. We create an integer program and use optimization routines that account for student submitted preference scores and site capacities. A GUI is also provided to help with data input and output for future use. Our results are guaranteed to be at least as good as the random lottery, and we see considerable improvement when assigning the Class of 2026.

1 Introduction

As part of the MD curriculum, medical students at the UNC School of Medicine are assigned to 1 of 6 branch campuses during the Application Phase. Each student is required to submit a ranking of each site as a 1st, 2nd, 3rd, 4th, 5th or 6th choice, with no ties allowed. During this process, students may also request to be placed together in pods of up to 3 students. From here out a pod will be defined as any group of students who must be placed at the same location, including a single student. That is, single students will be referred to as pods of size 1.

Previously, the assignment process was done using a random lottery. At each step, a random pod would be selected and placed in their highest-preference site that had capacity remaining. This would continue until all pods had been assigned to a site. However, the random lottery can still result in students receiving a 5th or 6th preference, which we seek to avoid whenever possible to increase the satisfaction of the students. This work optimizes the assignment process to minimize the number of students and pods being placed at the 5th or 6th preference by formulating a mathematical program and using an optimization solver. In addition, we provide an app to facilitate the use of the optimization code.

Using this optimization program results in assignments that will necessarily be at least as good (and commonly much better) than the random lottery. One such assignment is detailed in Table 1. This table represent an outcome of our optimization routine on data from 2022. The details of this table will be discussed further in Section 5, but for now we can see that students received only their 1st or 2nd choice. This is a great improvement on the lottery system, where we see that, on average there will be at least one student assigned to their 5th and 6th choices each.

Table 1: This set of coefficients further increases the objective value gained by 3rd and 4th preferences to further incentivize the solver to assign more pods to their top 2 preferences

Preference	Coefficient	Number of pods	Avg. number of pods in lottery
1	1	115	111.62
2	2	26	18.48
3	7	0	6.13
4	9	0	1.91
5	500	0	1.19
6	1000	0	1.67

In Section 2, we detail how data must be structured for input. Section 3 explains how to set up the app as well as run it. In Section 4, we explain the formulation of the optimization problem. In

Section 5, we display sample results from data for the Class of 2026 and compare them to results from a simulated random lottery.

2 Input Data

This section details the specific requirements for inputting data to the app. The input data is an Excel spreadsheet containing the preferences and size of each pod. Specifically, the columns must be in the following order:

Table 2: Required column specification for the input spreadsheet

Name	Description
Pod ID	Unique pod identifier
Asheville Central Charlotte Greensboro Raleigh Wilmington	Numbers 1 through 6 indicating pod preferences for each campus
Size	Size of each pod

Of the above column names, all but the ID column must be spelled exactly as specified (case-sensitive) as the software uses the column names from the spreadsheet to determine the names of the campuses where the pods are assigned. Only pods who have not submitted exemptions should be listed in this file. Any student who has an exemption requiring they be placed at a specific site need not be included in the optimization, as there is no decision to be made about their placement. Note that this Excel file does not contain information about site capacities. Inputting this information will be discussed in Section 3.

3 Shiny App

A Shiny app was created to facilitate the use of the optimization algorithms for placement of students to different sites.

3.1 App Set-up and Start-up

In order to use the app, you will need to first take the following steps:

1. Ensure R, RStudio, and Python are downloaded on your computer. R and RStudio can both be downloaded at <https://posit.co/download/rstudio-desktop/>, and Python can be downloaded at <https://www.python.org/downloads/>.
2. Ensure the app and all necessary code has been installed somewhere on your computer. This can be downloaded from the GitHub repository at https://github.com/plin97/STOR765_Spring2023_MedBranchAssignments. Download the whole repository to ensure all necessary

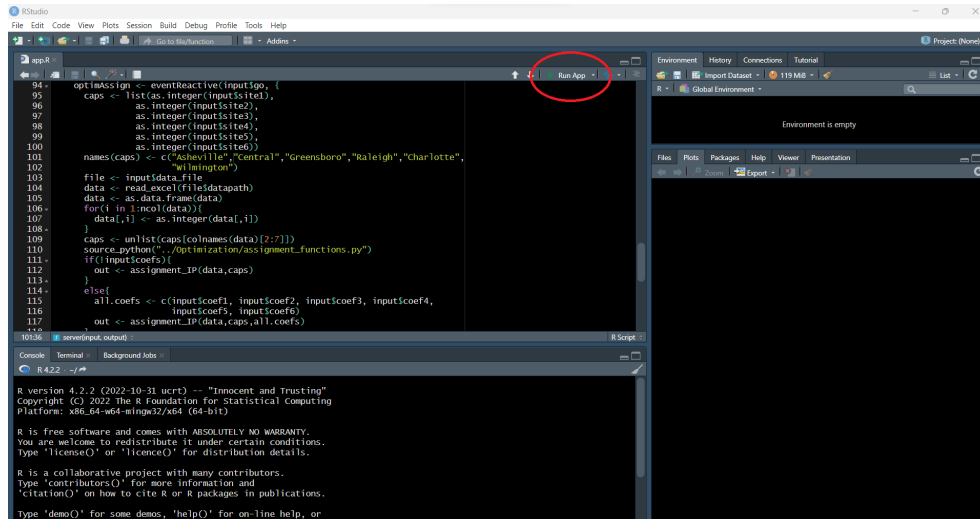


Figure 1: RStudio with the app open, and Run App highlighted

files are present. Do not change the directory structure (such as moving the files outside their respective folders or renaming folders).

3. Open RStudio on your computer.
4. In Rstudio, open the file app.R by selecting File, Open File, navigating to the App folder, and selecting app.R.
5. Press Run App in the top right corner of the app.R file. A screenshot of this screen with the Run Code button highlighted is provided in Figure 1.
 - If running the app for the first time, there may be additional installations happening in the background before the app will launch. If initial downloads are required, a prompt will appear in the lower left corner of RStudio, asking the user to type “Y” to approve the download or “N” to reject it. Our recommendation is to accept any installations automatically suggested by RStudio.
6. After a few seconds (during which the app is checking for additional packages needed for the optimization), a new window should open, which is the app interface.

3.2 Using the App

To use the app, follow the steps below:

1. Enter the capacities for each site as labelled. Note that the entered capacities should be post-exemption capacities. That is, if Central had a capacity of 78 and 2 students had exemptions requiring that they be placed at the Central site, then the entered number should be 76. See Section 3.3 for discussion of capacity overages.
2. Select the “Browse” button under “Choose Excel File”. A file explorer screen will appear and the user should select the Excel file containing the student preferences, ensuring that the file format follows the guidelines given in Section 2.

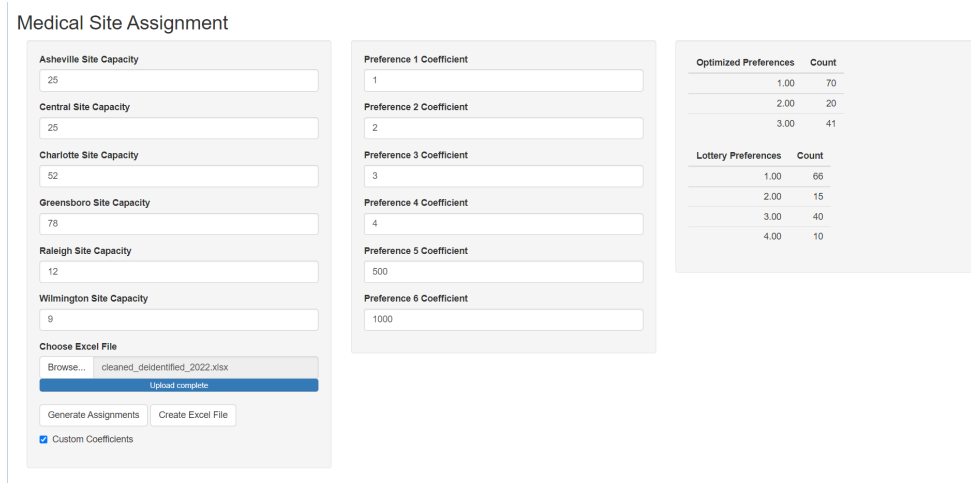


Figure 2: Example of App output with default values. Capacities were made-up for demonstration and data was the example data given at the beginning of this project.

3. If you wish to use custom optimization coefficients, select the checkbox underneath the “Generate Assignments” button, and a new column should appear allowing you to specify these values. In order for the optimization to work properly, the coefficients should be in increasing order from 1st through 6th preferences so that the larger coefficients act as a penalty for selecting a 6th preference.
4. Generate an assignment by pressing the “Generate Assignment” button. Statistics comparing the placements of students under the optimized regime against the random lottery will appear on the right side of the window. For reference see Figure 2.
5. Once the user is satisfied with the assignment as shown by the summary statistics, press the “Create Excel file“ to save the assignments for each student/pod. The Excel file will be located at source location of the app, with the name “OptimizeAssignment-Date-Time”, where Date and Time are the date and time the Excel file was generated. This naming procedure was adopted so that multiple assignments could be made without overwriting one another.

3.3 Capacity Overages

In certain years, there will be more students who need placement than the total capacity of all sites combined. In this case, any additional students will be placed at the Central site. For the purposes of the optimization routine, this is equivalent to increasing the capacity of the Central site to the minimum capacity needed to place all students. Our code automatically accounts for this overage. The user need not specify capacities greater than the total amount of students, as the optimization routine will automatically increase the capacity at the Central site. We note that the code does not notify users of capacity overages, so the user should make sure to check for this to make sure the result is as they expect.

4 Optimization

The optimization problem is formulated as an integer program (IP). Each pod is associated with six variables, corresponding to each of the six campuses. Each pod-campus variable can take value 0 or 1, with 1 meaning that this pod is assigned to that particular campus.

The objective function is a “score” for a proposed assignment, with pods being assigned to worse preferences adding much more to its value. By default, the values added are 1, 2, 3, 4, 500, and 1000 for 1st, 2nd, 3rd, 4th, 5th, and 6th preferences respectively. This set of values makes the solver much more unwilling to assign any 5th or 6th preferences due to the high coefficients of 500 or 1000 as it is trying to minimize this overall “score”. Modifying these values relative to one another can change the behavior of the solver, and may alter the number of pods assigned to their 1st, 2nd, 3rd, and 4th preferences, depending on the relative differences between the values. For example, increasing the values for 3rd and 4th preferences may reduce the number of pods assigned to a 3rd or 4th preference at the cost of some pods changing assignments from their 1st preferences to their 2nd or 3rd preferences.

The only constraints are that each pod can only be assigned to exactly 1 site, and that each site cannot exceed capacity. As such, the variables for each pod need to sum to exactly 1, and the sum of the sizes of the pods assigned to each site need to be at most the available capacity for that particular site.

Written out fully, the optimization problem is as follows:

$$\begin{aligned}
 & \underset{x}{\text{minimize}} && \sum_{\text{Pod-size combinations}} (\text{Preference score coefficient of a pod-site}) \times x_{\text{pod,site}} \\
 & \text{subject to} && \sum_{\text{pod}} (\text{Pod Size}) \times x_{\text{pod},s} \leq \text{Capacity}_s \text{ for each site } s, \\
 & && \sum_{\text{site}} x_{p,\text{site}} = 1 \text{ for each pod } p
 \end{aligned} \tag{1}$$

The solving of this IP is handled through a procedure called branch-and-cut. This algorithm attempts to find the optimal solution by solving a series of related problems to eliminate potential solutions that are known to be suboptimal.

4.1 Handling Pods

A key aspect is how the optimization procedure accounts for pods. When creating the optimization routine, two different procedures for dealing with pods were proposed:

1. Pod sizes would be accounted for in the objective function. Namely, our objective function would have become:

$$\sum_{\text{Pod-size combinations}} (\text{Preference score coefficient of a pod-site}) \times (\text{Pod Size}) \times x_{\text{pod,site}}$$

2. Pod sizes would not be accounted for in the objective function. Namely, our objective function would be as written in Equation (1).

On one hand, the first approach seems intuitively fair: a pod of multiple students being placed at a site with a low preference score would be worse than just one student being placed at a site with a low preference score, since more students would be at an undesired site. However, this would tend

to preferential placement for pods of multiple students. Consider the following example: There are two sites, A and B, and five students, who all prefer site A to site B, with two pods of two students and a lone student. If the capacity of site A is four and capacity of site B is three, then the solver would prefer to assign the two pods of two to site A and then assign the lone student to B. Under the objective function that accounts for pod size, this would be the optimal solution and the two pods are essentially receiving priority over the lone student. However, under the second objective function, the aforementioned solution would be considered equivalent to one pod of two and the lone student being assigned to A and the other pod to B, which is fairer to the lone student. In order to not favor pods of multiple students, formulation 2 was used.

5 Results

We run the optimization code on the 2022 data, with three different sets of objective coefficients to try to obtain more nuanced solutions. The results from each set are shown in Tables 3, 4, and 5. The tables begin by showing results using default values, and are adjusted as we tried to obtain better results. In the three tables, we can see that as the coefficients for 3rd and 4th preferences are increased, the optimal solution tends to place more pods at their top preferences due to 3rd and 4th preferences being penalized more.

We also simulate assignment as created by the random lottery 200 times using the same input preferences as a comparison to previous procedures and average the number of pods receiving each preference. These numbers are also reported in each of the tables. By subtly changing the coefficients used for the 3rd and 4th preferences, we end up changing the number of pods receiving their most preferred sites. Even with the default set of objective coefficients, our optimal solutions outperform the random lottery as we have fewer pods assigned to their 5th or 6th preferences.

Table 3: This is the default set of objective coefficients and primarily aims to minimize 5th and 6th preference placements and not making too much of a distinction between 1st through 4th preferences.

Preference #	Coefficients	Number of pods	Avg. number of pods in lottery
1	1	122	111.62
2	2	17	18.48
3	3	1	6.13
4	4	1	1.91
5	500	0	1.19
6	1000	0	1.67

Table 3 shows the result of using the default coefficients. We note that this result is already better than the average lottery as we have more 1st choices and fewer 6th, 5th, 4th, and 3rd. However, we need not be satisfied by these results. We can try to tweak the coefficients so that pods who have been assigned their 4th preference will get a better assignment.

Table 4 shows the result of increasing the 4th preference coefficient to disincentivize placing students at their 4th choice. We notice that, although no student is being placed at their 4th preference, there is a cost of the cost pods who were previously placed at their first choice now

Table 4: These coefficients penalize 4th preference placements to try to get more pods placed at one of their top 3 preferred campuses.

Preference	Coefficients	Number of pods	Avg. number of pods in lottery
1	1	120	111.62
2	2	18	18.48
3	3	3	6.13
4	7	0	1.91
5	500	0	1.19
6	1000	0	1.67

being placed at their second or third preferences. Ultimately, it is up to the user to decide which outcome they would prefer. Finally, we can continue to further adjust the coefficient values to see if we can get all students to be at their first or second preference.

Table 5: This set of coefficients further increases the objective value gained by 3rd and 4th preferences to further incentivize the solver to assign more pods to their top 2 preferences

Preference	Coefficient	Number of pods	Avg. number of pods in lottery
1	1	115	111.62
2	2	26	18.48
3	7	0	6.13
4	9	0	1.91
5	500	0	1.19
6	1000	0	1.67

Finally, Table 5, which is a copy of table 1. Again, we see we have improved the assignment in the sense that every student gets their first or second choice, at the cost of fewer 1st choices. We note that we increased the value of the coefficient for a 3rd choice to 7, which meant we also needed to increase the coefficient for a 4th choice to avoid 4th preference being better than 3rd preferences. The value of the coefficients is only important relative to one another, and the most important aspect is that lesser desired placements have higher coefficients.

6 Software

R and RStudio were used to create the app. The R packages used were tidyverse, shiny, reticulate and readxl. Python was used to program the optimization routine and uses Numpy, pandas, and Python-MIP. The optimization solver (included as part of Python-MIP) is Cbc ¹.

¹<https://github.com/coin-or/Cbc>