# Operations Research II Final Project: Optimizing Big/Little Pairs

*by: Alice Liang, Cyrus Castellanos, Shreya Dasari, and Shreya Jagnade*

## Introduction

Each semester at Carnegie Mellon (and other colleges around the country), fraternities and sororities across campus recruit a new class of members. A critical part of the new members' integration into the chapter is being assigned a sorority/fraternity "big". A "big" is an older member of the chapter who guides their "little" as they become accustomed to Greek Life. "Big/little" pairings are based on how the set of bigs ranks each of their top choices for littles and how the set of littles ranks each of their top choices for bigs. Usually, every person ranks their top 5 choices from 1-5, with 1 being their top choice. However, this is not always the case. For example, if a certain big only wanted to rank their top 3 choices instead of 5, they are usually permitted to do so.

After each little and each big submit their ranking, the Greek organization tries to pair each little with the big that would be the best fit for them. Every little/new member MUST have a big, but it is not guaranteed that every big gets a little. It is also possible for a big to have two littles (twins).

One important aspect about big/little pairing is that the little's preferences have a larger weight than the big's preferences. To capture this in our ranking algorithm, we make sure to consider the littles' rankings first and match them with their top available big rather than considering the bigs' rankings first.

Our approach to algorithmically make big/little pairings was to apply the Stable Marriage Problem and Gale Shapely Algorithm to an anonymized dataset of big/little rankings.

## Description of Dataset

We applied the optimization problem to a dataset of big/little rankings. Every potential big and little must attend a minimum of 5 "big/little dates" with a member from the opposite set before ranking. The big/little dates are randomly assigned. Out of their 5+ big/little dates, each big and little ranks their top 5 choices. They are permitted to rank less than 5 people, however this is discouraged. A rank of 1 is their top choice, while a rank of 5 is their

bottom choice. We want to optimize pairings such that no big and little would mutually prefer to be with each other over their currently assigned match. The dataset is the rankings of all of the bigs and littles. All data is anonymized and randomized.

## Methods

The stable marriage problem looks at N men and N women. Each of the men and women rank members of the opposite sex by preference. We aim to create marriages between the set of men and women such that no two individuals would prefer to be with each other over their current assigned partners. If no pair like this exists, we consider the marriage stable.

Consider two men, Abe and Bob, and two women, Carol and Dolly. Consider the following scenario:

Abe prefers Carol over Dolly

Bob prefers Carol over Dolly

Carol prefers Abe over Bob

Dolly prefers Abe over Bob

Note that both Abe and Carol mutually prefer each other, while Bob prefers Carol and Carol prefers Abe. Note that the pairings (Abe, Dolly), (Bob, Carol) are not stable as both Abe and Carol would rather mutually be with each other. Note that the pairings (Abe, Carol), (Bob, Dolly) are stable because even though Bob prefers Carol and Carol prefers Abe, this is not a mutual preference. Since Abe and Carol would prefer to stay within their assigned couples and not with anyone else, these pairings create a stable marriage.

Now, let's consider the Gale-Shapley algorithm:

```
def wPrefersM1OverM(prefer, w, m, m1):
    N = len(prefer[0])
    for i in range(N):
        if prefer[w][i] == m1:
            return True
```

```
        if prefer[w][i] == m:
            return False
    return False


# Implements the stable marriage algorithm
def stableMarriage(prefer):
    N = len(prefer[0])
    wPartner = [-1] * N  # Stores women's partners
    mFree = [False] * N  # Tracks free men
    freeCount = N

    while freeCount > 0:
        m = next(i for i in range(N) if not mFree[i])
        for i in range(N):
            if mFree[m]:
                break
            w = prefer[m][i]
            if wPartner[w - N] == -1:
                wPartner[w - N] = m
                mFree[m] = True
                freeCount -= 1
            else:
                m1 = wPartner[w - N]
                if not wPrefersM1OverM(prefer, w, m, m1):
                    wPartner[w - N] = m
                    mFree[m] = True
```

```
                    mFree[m1] = False
    return wPartner
```

Let the input be a 2N x N matrix, where N is the number of men and women. So, in rows 0-N-1, we have the men's preferences, each as a 1-D N list. In rows N-2N-1, we have the women's preferences.

Consider the function wPrefersM1OverM, which indicates true when a woman prefers man m1 over man m, else false. We loop through the women's preference list. If m1 appears first, this indicates that she prefers him, and we return false. If m appears first, this indicates that she doesn't prefer m1 over m, and we return false. If neither m or m1 appears, we return false as the default case.

Consider the final stableMarriage preference which takes in the 2N x N matrix. We initialize a partner array of length N to store the women's partners. For now, we initialize all the values to -1. We initialize a free array of length N to store which men are free. For now, we initialize all the values to false.

While the number of free men remaining is greater than zero, we pick the next free man, m. We go through that men's preference list, and we denote w as the woman the man prefers. If the woman is free, then the man and women become engaged, we assign m to w, mark m as engaged, and indicate there is one less free man. If the woman is engaged to some m1, check if she prefers m over her current partner m1. If she does, then the man and women become engaged, we assign m to w, mark m as engaged, and mark m1 as free. Finally, we return the matching which is an array of length N.

The Stable Marriage Problem provides a general framework for the kind of problem we are interested in solving for Big/Little pairings. However, our problem has "looser" constraints, such that rather than having each big only be paired with one little, a big can have up to as many littles as they would prefer. Additionally, the preference lists of bigs and littles do not need to rank every single other little and big respectively, only their top choices. While stable marriages fall within the set of one-to-one matchings, Big/Little pairings fall within

the set of many-to-one matchings. A common version of the many-to-one matching problem with under-full ranking lists is known as the University Admissions problem.

In the classical University Admissions problem, every college can admit multiple students up to their capacity, while each student is admitted to at most one college. Let S be the set of colleges and C be the set of students. Every college has a strict ranking over students and every student has a strict ranking over colleges. These rankings do not need to include every single student for college lists nor every single college for student lists, so it is possible that some colleges may not get filled and some students may not get admitted. Regardless, the sets of admitted students for a particular college are disjoint such that no student is admitted to more than one college.

The idea of an optimal matching in the University Admissions problem is similar to that of the Stable Marriage problem. We still seek to achieve stability in whatever matching we end up with, such that:

Every matched pair of student and college, (s, c), must prefer each other to being unmatched

There must be no pair, (s, c), such that both of the following hold:

- s is either unmatched or prefers c to its current match
- c either has spare capacity or prefers s to its lowest ranked admitted student

Similar to the Stable Marriage problem, a stable matching of students with colleges always exists, regardless of the number of students or colleges, or the length of each student and college's ranking lists. With this framework, we can easily translate the idea of colleges to bigs and students to littles while maintaining the same idea of stability. Thus, any algorithm that produces an optimal matching for the University Admissions problem must also be able to produce an optimal matching for our Big/Little problem. Let us consider one such algorithm:

```
gale_shapley_modified(little_prefs, big_prefs, capacities):
    initialize matches and proposal counters
    while there are free littles:
        little proposes to next preferred big
        if big has capacity:
            accept little
        else:
            if little is preferred over worst assigned:
                swap little in, free worst little
            else:
                little remains free
    return little_match, big_match
```

This pseudo-code algorithm describes the modification to the Gale-Shapley algorithm we need in order to solve a many-to-one matching problem with under-full ranking lists. We will apply this algorithm to Python on our dataset to produce an optimal stable matching of bigs and littles. The results are displayed in the next section.

## Results

Our data set contains rankings for 28 Bigs and 29 Littles. Each Big and Little ranked up to 5 Littles or Bigs respectively, although some ranked less. Bigs also had a certain capacity for how many Littles they were willing to be matched with, with most being willing to take on up to 2.

We took our dataset and passed it into our version of the University Admissions algorithm constructed in Python. The algorithm had two outputs, one being a list of each Big and their Littles, and the other being a list of each Little and their Big. Based on the results shown in the graphics above, every Little was matched with a Big, although some Bigs were not matched with Littles. It should be noted that, similar to matchings constructed by the

Gale-Shapley algorithm for the Stable Marriage problem, the stable matching our algorithm found is not necessarily unique. There may be other matchings that could be constructed simply by inputting our data in a different order that are just as optimal by our definition of stability.

The the pairings generated from the algorithm for our particular dataset are listed on the following pages.

# Big Matchings

Big 1 : Little 1

Big 2 : Little 20

Big 3 : Little 13, Little 23

Big 4 : Little 19

Big 5 : Little 4

Big 6 : Little 17

Big 7 : Little 3, Little 18

Big 8 :

Big 9 :

Big 10 : Little 8

Big 11 : Little 15

Big 12 : Little 6, Little 29

Big 13 :

Big 14 :

Big 15 : Little 7

Big 16 :

Big 17 : Little 12, Little 22

Big 18 :

Big 19 : Little 9

Big 20 :

Big 21 : Little 5

Big 22 : Little 11, Little 14

Big 23 :

Big 24 : Little 16, Little 21

Big 25 : Little 10, Little 26

Big 26 : Little 27, Little 28

Big 27 : Little 25

Big 28 : Little 2, Little 24

# Little Matchings

Little 1 -> Big 1

Little 2 -> Big 28

Little 3 -> Big 7

Little 4 -> Big 5

Little 5 -> Big 21

Little 6 -> Big 12

Little 7 -> Big 15

Little 8 -> Big 10

Little 9 -> Big 19

Little 10 -> Big 25

Little 11 -> Big 22

Little 12 -> Big 17

Little 13 -> Big 3

Little 14 -> Big 22

Little 15 -> Big 11

Little 16 -> Big 24

Little 17 -> Big 6

Little 18 -> Big 7

Little 19 -> Big 4

Little 20 -> Big 2

Little 21 -> Big 24

Little 22 -> Big 17

Little 23 -> Big 3

Little 24 -> Big 28

Little 25 -> Big 27

Little 26 -> Big 25

Little 27 -> Big 26

Little 28 -> Big 26

Little 29 -> Big 12

## Conclusion

We can argue that the above results demonstrate that the University Admissions algorithm, a variation of the Gale-Shapley algorithm, optimally pairs Bigs and Littles for the purposes of sorority and fraternity life. This algorithm can be used to assist the Big Little pairing process in the future. In some sense, the process can suffer because it is largely dictated by the hands of sorority and fraternity leadership, for whom it can be hard to remain unbiased. The use of this algorithm could make this long, grueling process where it can be hard to make everyone happy, more objective and efficient. However, while we can take this a step further and advocate for more automated and algorithmic social pairings throughout society, as currently most grouping and pairing rely on personal decision making, we must examine the core measurement of compatibility - the ranking scheme. The reliability of this algorithm rests heavily on the fact that an initial coffee chat between potential Big and Little, ranging from 30 minutes to a maximum of 2 hours, allows each Big and Little to accurately assess their compatibility with the other person. Of course, while the ranking scheme is flawed, it cannot be removed, but if we replace the Big Little pairing process that requires hours of work for Greek life leadership where they take into account personal relationships, future COB opportunities, amongst other insider information that the algorithm cannot possibly know on top of these rankings, we directly oppose the goal of sorority and fraternity life: to foster community. This pairing system is successful because it is a deeply personal process. Ultimately, while the University Admissions algorithm can be used to assist Big Little pairing as we have demonstrated its correctness in this usage, it should not entirely replace this fundamentally human process that has long created lasting friendship and mentorship.

## References

References GeeksforGeeks. "Stable Marriage Problem." GeeksforGeeks, 20 July 2013, www.geeksforgeeks.org/dsa/stable-marriage-problem/. Accessed 30 Nov. 2025.

Manlove, David F, et al. "Hard Variants of Stable Marriage." Theoretical Computer Science, vol. 276, no. 1-2, Apr. 2002, pp. 261–279, https://doi.org/10.1016/s0304-3975(01)00206-7. Accessed 29 Aug. 2022.

Michael Levet. "Algorithmic Game Theory- College Admissions Problem." Wordpress, 10 Jan. 2016, michaellevet.wordpress.com/2016/01/10/algorithmic-game-theory-college-admissions-problem/. Accessed 30 Nov. 2025.

Sethuraman, Jay, et al. "Many-To-One Stable Matching: Geometry and Fairness." Mathematics of Operations Research, vol. 31, no. 3, Aug. 2006, pp. 581–596, https://doi.org/10.1287/moor.1060.0207. Accessed 13 Sept. 2022.

"Understanding the Big and Little Relationship." TheSororityLife.Com, 4 Sept. 2024, thesororitylife.com/sisterhood/learn/understanding-the-big-and-little-relationship/.

Taynor, Heather. "What Is Big and Little in Greek Life? A Detailed Guide." Clock Tower Higher Education - Higher Education, Simplified., 27 Oct. 2025, clocktowerhighered.com/campus-life/what-is-big-and-little-in-greek-life/how-the-big-and-little-process-works-in-greek-life.