Project 2: Final Paper
The Stable Marriage Problem

Kristen Lawler

May 8, 2017

1 Abstract

The stable marriage problem addresses the following issue: suppose you have a population that can be split in half by some defining characteristic (for example gender). If each member of this population were to have a preferential list of possible mates, then how could they all be paired together so that the resulting matchings are stable?

A common solution to this problem follows an iterative process of proposals and acceptances or rejections. In order to model this process in JAVA, an Individual object was created and utilized to represent each individual in the population. All were split into two disjoint array lists to represent the division of the population and for each individual, aside from the user, a random list of preferential mates was generated. Then, an iterative process was used so as to pair up all individuals while considering these lists of preference.

Though many solutions do exist to this problem, such as the Gale-Shapley algorithm, most are traditionally programmed in C++ or Python, making this solution unique in its own right. This problem is particularly useful since its solution can be applied to many real life situations. These include residency assignments in hospitals, college housing assignments, college acceptances, and on-line dating applications.

2 Introduction

A solution to the stable marriage problem must result in a “stable” matching of all individuals within the population. In order to achieve this stability, we must have either that each individual is paired with their preferential mate, or that the mate they prefer is paired with someone higher on his or her own preference list. In order to ensure this is achieved, we select one “side” of the population to be the proposers. Beginning with their most preferential mate, each individual on the proposer side of the population offers to be matched to a mate on the other. The mate either accepts the proposal or rejects it based upon their own preferences. This process repeats until all individuals have a mate.

The motivation of this project is to utilize JAVA so as to emulate the previously mentioned process. This methodology of matching has been mathematically proven to result in stable pairings so long as the population size is even, and no one individual is “hated” by all potential mates. This specific program allows for the user to act as a member in the population. As such, it takes the user’s input for a list of preferences. It also takes in a user input for the number of final pairings.

3 Detailed System Description

The first step in writing a program that will address and solve the stable marriage problem was to create an object that could be used to represent each individual in the population. The Individual object was therefore created. Each Individual has an I.D attribute. This is used to specify Individuals during the proposal process. Individuals also contain an array which holds their preferential list of mates. For all Individuals aside from the user, this list is randomly generated using a method called permutation. These objects contain several other attributes which are utilized during the matching process to indicate whether an Individual has been paired or not, to store the I.D. of the Individual when paired with them, and to count the number of proposals each “suitor” has given. The Individual object was used to

Suitor
identity: int matches: int rank: int[] onHold: int taken: boolean waitList: int[] numProps: int
Individual(x: int, m: int) Individual(int x, int m, int[] p) permutation(int): int[]

generate both sides of the population (the "proposers" and the "deciders").

After getting the user's input for how many matches there should be in the population (essentially one half of the total population size) which we will refer to as n, the program generates two arrays lists of the specified size holding n many Individual objects each with a different I.D. (0 to $n-1$). These represent the separated halves of the population. The user has a default I.D. of 0. The object associated with the user is placed as the first element of the "decider" array list. Since we allow for the user to specify their own list, there are two different constructors that can be used to create an instance of the Individual object. One allows for an array input and one does not.

After receiving all user inputs, the Gale-Shapley algorithm is applied in order to produce a stable pairing of individuals prom both sides of the generated population. Starting with their most preferential candidate, proposers offer to be matched with one decider. As they propose,individuals are added by their I.D. to the appropriate decider's waitList attribute. Once all proposers have had the chance to propose, the deciders look through the Individuals on their waitLists and select the most preferential candidate based upon their own list of preference. All proposers who were not selected then propose to their next most preferential candidate and so on. This process repeats until all proposers and deciders have been matched. In the beginning of the program, each decider and proposer's list of preference is given. WaitLists as well as matchings are then outputted with each round of proposals and acceptances. The final stable pairing of all Individuals is the final output of this program.

4 Requirements

The problem that this program needs to address is how to pair individuals in a stable fashion. To increase flexibility and applicability, this program allows for a user input, enabling the user to be a part of the population that this program will match. This program will also allow for the user to determine how many final matching there will be, and thus how many individuals there should be in the population. Once getting the user's data, this program is required to access the first ranked Individual on each proposer's list of preference. The proposer must then be added to the corresponding decider's waitList, and the proposer's numProps attribute is increased by 1 to ensure that if rejected, the proposer will offer to be matched with its next most preferential candidate and so on (numProps is used to index through the preferential list). Next, this program must find the highest ranking proposer on each waitList based upon the decider's preference, place them "onHold" as a potential final match for the decider, and switched the proposer's taken attribute to true. This process needs to repeat until we have that each decider has a proposer onHold, which signifies that all individuals have been matched.

It is important to note that there is the possibility that a decider who has a proposer on hold receives another proposal. For that reason, at the conclusion of each round, the deciders who have a proposer on hold have their proposer added to their waitList for the next round. All other deciders have their waitLists wiped and ready to hold any possible proposals in the following round. If a new proposer is selected and put on hold, the previous match will have its "taken" attribute switched back to false and will make a proposal in the coming round.

5 Literature Survey

While the theory of this algorithm is a famous solution to the stable marriage problem, there do exist others. For example, Gusfield and Irving offered a similar but alternative algorithm that serves as a solution. Although the algorithm used in the project itself is not entirely unique, this representation of the algorithm is. Most solutions or representations of the Gale-Shapley algorithm are coded in C++ and python. My representation is coded in JAVA and takes an object oriented approach making it vastly different than previous coded solutions that I have seen.

6 User Manual

When prompted to enter the number of matchings desired in the final outcome, the user must input some positive integer, n. When then prompted to input a preference, the user must enter numbers 0 to $n-1$ in the order that they desire (most preferential to least preferential). Each integer refers to a proposer for the user to be matched with. It is important to note that the user must account for every proposer when giving his/her list of preference.

The program will then output all proposals, all updates to each decider's waitList, all matchings produced in each round, and will conclude with the final stable pairing.

7 Conclusion

The stable marriage problem addresses how to match up individuals in a population based upon all of their preferences. Utilizing the created Individual object, this project replicates the process known as the Gale-Shapley algorithm that serves as a mathematically proven solution to the stable marriage problem. The known algorithm is so highly praised because it not only generates stable pairings, but it proves that for given population, a stable pairing must exist. The solution and problem can be applied to many different areas. This specific representation of the Gale-Shapley algorithm has the ability to generate a population and preferences based upon user inputs, thus this program can easily be extended and utilized in other areas of application including other algorithms that are concerned with matching and preference.

8 Bibliography

D. Gale and L.S. Shapley, "College Admissions and the Stability of Marriage," American Mathematical Monthly 69 (1962), pp. 9–14.

Gusfield, Dan, and Robert W. Irving. The Stable Marriage Problem: Structure and Algorithms. Cambridge Mass.: MIT, 1989. Print. Foundations of Computing Ser.

Hunt, William. "The Stable Marriage Problem." West Virginia University, Morgantown.

Sahai, Anant. "The Stable Marriage Problem: An Application of Induction in Understanding Algorithms." University of California, Berkley.