

# COMP 303: Project 2: Stable Marriages

Vivian McPhail

199200332

## 1 Design

Two basic design techniques were used for this assignment, a greedy method, and dynamic programming. The dynamic programming element was the use of an array to store the the current top choice of each candidate. A greedy choice was used when selecting a marriage from a preference chain.

## 2 Algorithm

The algorithm is as follows: first, build a preference chain. Select a candidate,  $C$ , at random, then find that candidate's preferred spouse  $S$ . If this person is married, then remove them from the list of preferences. This marriage is ranked by combining the preferences of both candidates. If this marriage is the highest ranked marriage seen in the preference chain, remember it. If both candidates are each others' first choice then the rank will be  $0 = (1 - 1) + (1 - 1)$ . The rank a particular marriage receives depends upon the set of single candidates, so that if a candidate's first two choices are already married, the third person on their list has a rank of one. After ranking the marriage, the candidates are added to the set of active candidates. If the preferred spouse is already in the set of active candidates, then the chain has developed a cycle. Otherwise, the spouse is used to repeat the process until a cycle is formed.

Once a cycle has been obtained, discard the (possibly empty) head which is not part of the cycle, then, from the cycle choose the highest ranked marriage, and add the two candidates from this marriage to the married set. Now, build a new chain and repeat the process.

## 3 Correctness

The goal is to avoid any unstable marriages. Now, by building a preference chain, we are assuring that there will be no unstable marriages, since all candidates in that chain are paired forward with their favourites. If a candidate's choice is also the top preference of the spouse then this is an optimal marriage. Otherwise, the only possible problem is the reciprocal marriage, which will not

necessarily be the first choice. But even if the spouse  $S$  does not prefer  $C$ , we know that their preference will not be reciprocated, since that person's first choice is already in the preference chain. By always choosing the highest rank marriage, we will achieve a global optimum of preferences, even though we never did any global comparisons (unless the chain is of length  $2n$ ).

## 4 Analysis

For  $n$  men and  $n$  women we have to build  $n$  preference chains, since each time we build one we select one marriage. Note that a more clever algorithm than the one used here can use portions of previously built preference chains to save on computation. Each preference chain could potentially include all the remaining candidates, so that the first round could be  $2n$  long, the second  $2n - 2$ , the third  $2n - 4$ , and the last 2 long, so that the total would be  $2(\frac{n(n-1)}{2}) = n(n-1)$  for a worst-case complexity of  $O(n^2)$ . Now, if each candidate's first choice is also their choice's first choice, then each chain will be 2 long, with  $n$  chains we have a best-case complexity of  $\Omega(n)$ , so we have an overall complexity of  $\Theta(n^2)$ .

## 5 Data Structures

An array of lists was used to store the preferences. Sets implemented as lists were used to store the set of married and active candidates.

## 6 Implementation Differences

An extra degree of complexity was introduced in the actual implementation for two reasons, Whenever a spouse was found for a candidate that spouse had to be checked against the married set, with this implementation, an  $O(n)$  operation, also, when a marriage was ranked, the list of preferences has to be searched for the rank, which is  $O(n)$ . These complexities are additive with respect to each other, but are performed at each step of building the chain, so that we have worst case complexity of  $O(n^3)$ , the best case does not change.

More efficient implementations of the data structures would restore the theoretical complexity. For example, since the number of candidates is known, the sets could be implemented as bit sets, and the array could have been a map from candidate to candidate to preference, so that each ranking operation is constant time.

## 7 Testing

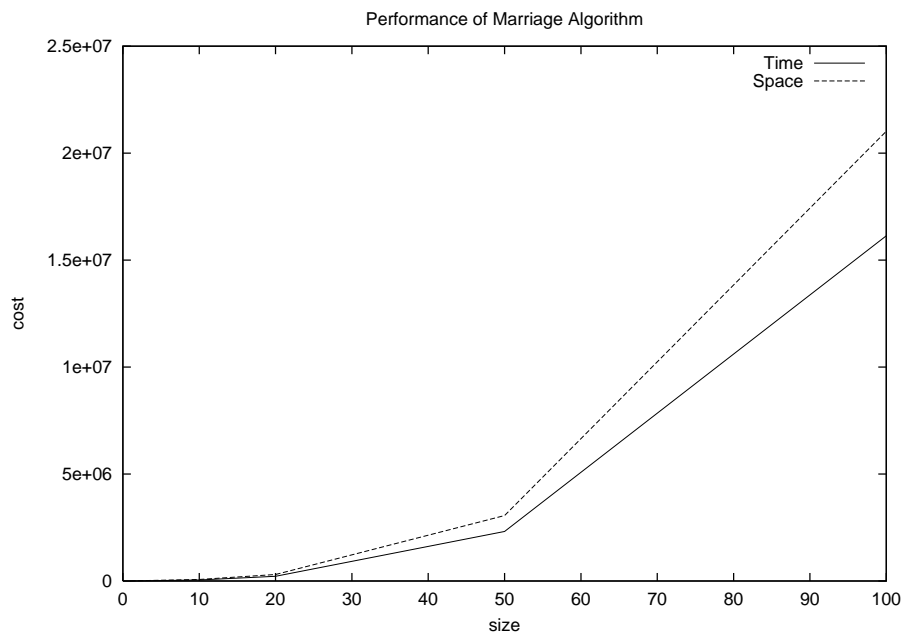
Three basic cases were tested. Also, testing was done on preference sets in which every one had the same preferences, which should result in an exactly average case result.

## 8 Performance

The number of reductions performed by the haskell interpreter was used as a proxy for time, the number of cells was used as a proxy for space:

<i>size</i>	<i>time</i>	<i>space</i>
1	1878	3052
10	47893	68715
20	219513	300965
50	2318373	3061715
100	16136473	21022967

This looks like



## 9 Comparison

The experimental results indicate a polynomial running time algorithm, which was expected.