Stable Matching Report

Andreas Skovdal, Laura Vikke Mårtensson, Mads Høgenhaug, Marcus Friis & Mia Pugholm

September 7, 2022

Results

Our algorithm has provided the expected results.

Implementation details

Overall, the stable matching algorithm is implemented as in accordance to the Gale–Shapley algorithm[1]. The algorithm runs in $O(n^2)$. This is achieved by using data structures with minimal time complexity based on the needed functionalities.

[1] We used the pseudocode as reference from [KT 1.1].

All available men with no matches are stored in a list, `free`. This list is handled with stack discipline, using only the pop and append to get, remove and add men. This allows us to find a free man who has not proposed to every woman in time $O(1)$, by popping the last element from the `free` list. Likewise, we can also add men back to the list with `append` in time $O(1)$, when a man is freed up.

The men's and woman's preferences are stored in two separate dictionaries `proposers` and `rejecters`, mapping each person to a preference linked list. The linked list allows for accessing and removing the first element from preferences in $O(1)$[2]. This is necessary to keep track of who he has proposed to.

[2] The code is implemented in Python. All python time complexities is referenced on `https://wiki.python.org/moin/TimeComplexity`.

For keeping track of our matchings, we create a hashmap `matchings`. We initialize all women in `matchings` with a `None` value. This allows us to use the given women as a key to lookup her proposal status. This is done in constant time, due to lookups in hashmaps being $O(1)$.

When a man is proposing to a woman who is already engaged, we must be able to compare to men's ranking in constant time. To do this, we construct an n x n matrix keeping track of women's preferences. Constructing this requires looping through women and women's rankings, which results in $O(n^2)$ time complexity. This data structure allows for constant time lookup of the ranking with a simple lookup in the matrix `ranking[w,m]` at the cost of additional initialization time.