Liam Tolkkinen

Krishnaprasad Thirunarayan, Ph.D.

Assignment 01

12 February 2024

Gale-Shapely Algorithm Theoretical Analysis

INTRODUCTION

The Gale-Shapely Algorithm provides a stable matching for a listing of men and women. A stable matching ensures that there are no pairs M-W such that M and W prefer each other over their current partner. This is a modification to the algorithm introduced in class where there were an equal number of men and women. This algorithm supports any number of men with any number of women such that the number of men and the number of women are independent.

ORIGINAL PSEUDOCODE

```
function GaleShapley(M, W):
    Initialize all men and women as free
    while there exists a free man m who still has a woman w to propose
to:
        w = m's highest ranked woman to whom he has not yet proposed
        if w is free
            (m, w) become engaged
        else if w prefers m to her current partner m':
            m' becomes free
            (m, w) become engaged
        else:
            m remains free
    return the list of engaged pairs
```

UPDATED PSEUDOCODE

```
Function propose(M, W): //this is a recursive function
if w is free:
accept (W is now M’s partner and likewise)
return true to exit function (base case!)
    else:
        Check if w’s current partner is better than M
        If current partner is better:
            Reject!
            return false to exit function (base case!)
        else:
            Accept M as new partner
            Break up with current partner (M’)
            //recursion starts:
            For woman in M’.preferences:
            propose(M’, W) //until returns true (Match!)
            return true //this is a base case for the recursion
```

```
function GaleShapley(M, W):
    Initialize all men and women as free
    Add men and women to an array
    Initialize their preferences as an array
    for man in men array:
        For woman in man’s preference list:
            propose(man, woman) //until returns true (match!)
```

```
DONE!
```

Since propose(m,w) is a recursive function, it does all the work of matching men who were broken up with by their previous partners. **This works for unequal numbers of men and women because the recursive function loops through until the severed man has proposed to every woman on his list. In the case that there are more men than women, there will be a set of men that are rejected by every woman on their list. Whether or not this is after the break up or before.**

In the case that there are more women then men, there will be a set of women who are never proposed to. This is always true because it is known that a woman never becomes single after accepting a proposal, she can only accept new proposals if it suits her.

COMPLEXITY ANALYSIS

RUNTIME

Guide:

N = #men

M = #women

Initialization of arrays of men and women: O(M+N)

For each man and women, initialize preferences...

Therefore, total initialization = O(M*N + N*M) = O(2M*N) = O(M*N)

Every man proposes to every woman once (worst case scenario) therefore proposal = O(N*M)

During matching: woman W needs to traverse preference list.

This happens for every woman, therefore matching = $O(M^2)$

The runtime complexity becomes:

$O(N*M) + O(N*M) + O(M^2) = 2*O(N*M) + O(M^2) =$ **$O(N*M) + O(M^2)$**

This is where it becomes complicated...

In cases that $N \geq M$: $O(N*M) \geq O(M^2)$.

But when $N < M$: $O(N*M) < O(M^2)$

However, we can generalize and say that the runtime complexity is $O(M^2)$

SPACE COMPLEXITY

Guide:

N = #men

M = #women

Initialization of men and women lists: O(N + M)

For each man and woman, make a list of preferences:

Therefore, the initialization becomes O(N*M + M*N) = **O(N*M)**

Recursion stack: the recursion stack will grow to a maximum of N frames. If each stack frame is of size K, then the recursion stack has a space complexity of O(kN) = O(N).

The total space complexity of this modified Gale-Shapely algorithm can be expressed as:

O(N*M) + O(N) ==> O(N*M)