CS 6901 (Applied Algorithms) – Lecture 2*

Antonina Kolokolova

September 15, 2016

1 Stable Matching

Recall the Stable Matching problem from the last class: there are two groups of equal size (e.g. men and women, interns and companies, grad. school applicants and potential supervisor), where every member of one group has a "preference order" of all members of the other (e.g., every company has a arranged the list of all potential interns in order of decreasing desirability, and every intern has a similar list of all companies from one she likes the most to the least).

Today we will talk about the (Nobel-prize winning) algorithm that solves this problem, prove its correctness and discuss its properties.

Let us follow the textbook and tradition in calling the two sides "men" and "women". The algorithm, due to Gale and Shapley, works as follows.

StableMarriage(n, M, W)

// Here M is an array of men's rankings, and W is an array of women's rankings.
1 Initially all m_i and all w_j are free. Initialize the set S of matched pairs to $\emptyset$.
2 **while** there is a free m_i
3 let w_j be the highest-ranking woman in m_i's list whom m_i has not approached yet
4 **if** w_j is free, add (m_i, w_j) to S
5 **elseif** w_j prefers m_i to its current match m_k , remove (m_k, w_j) from S and add (m_i, w_j) to S
6 **elseif** w_j prefers her current match, do nothing

7 **return** S

*The following topic is adapted from Kleinberg/Tardos "Algorithm design"

To see how the algorithm works, consider the following examples.

Example 1 Let our input be as follows.

$$m_1 : w_1, w_2 \qquad w_1 : m_2, m_1$$
$$m_2 : w_1, w_2 \qquad w_2 : m_2, m_1$$

That is, both men prefer first woman over second, and both women prefer the first man. Here is one possible execution of the algorithm. At the start, $S = \emptyset$ and all of m_1, m_2, w_1, w_2 are free.

1) m_1 proposes to w_1. Since she is free, she accepts. S becomes $S = \{(m_1, w_1)\}$.

2) m_2 proposes to w_1. Although she is engaged to m_1, since she prefers m_2 she switches to him. Thus, m_1 becomes free and S becomes $S = \{(m_2, w_1)$.

3) Now, m_1 is free again, so he proposes to the next woman in his list, to w_2. Since she is free, she accepts. So $S = \{(m_1, w_2), (m_2, w_1)\}$.

4) Every man is engaged now, so the algorithm terminates.

A different run of the algorith on this input could have m_2 doing his proposal before m_1. In that case, after both the first and the second rounds, $S = \{(m_2, w_1)\}$, since when m_1 proposes to w_1 she is just rejects him.

We will show soon that no matter in which order men propose, the algorithm always produces the same matching.

Example 2 Now consider the other example we have looked at before.

$$m_1 : w_1, w_2 \qquad w_1 : m_2, m_1$$
$$m_2 : w_2, w_1 \qquad w_2 : m_1, m_2$$

That is, men's preferences are a reverse from women's. Recall that there are two possible matchings in this case, one where men get their first choice and a different one where women do.

Here is a possible run of the algorithm.

1) m_1 proposes to w_1. Since she is free, she accepts. S becomes $S = \{(m_1, w_1)\}$.

2) m_2 proposes to w_2. She is also free, so she accepts. $S = \{(m_1, w_1), (m_2, w_2)\}$.

3) Every man is engaged now, so the algorithm terminates.

Here, n rounds were enough to construct a matching, since no man was ever rejected.

Notice that in this case the matching the algorithm constructed favours men (the side that proposes). Indeed, it can be shown that this is always the case.

1.1 Correctness proof

The algorithm worked on those two examples, but how do we know that it will always terminate, and that when it terminates, it will always construct a stable matching. It is not even clear apriori that it will construct any perfect matching.

Theorem 1 *The Gale-Shapley algorithm for stable matching always terminates and produces a stable matching on termination.*

Proof: First, let's argue that the algorithm will terminate. For that, we would like to have some quantity (measure of progress) that will strictly decrease at every round with the property that when it reaches 0, the program stops. (Equivalently, we can take an increasing quantity which is guaranteed not to grow larger than a given bound.)

Here, let's (deviating a little from the KT book) take as such progress measure the number of pairs (m, w) such that m has not proposed to w so far. Originally, there are n^2 such pairs. At every round some man proposes to a woman he has not contacted before; this eliminates one pair from the set. Notice that a man never proposes to a woman twice; if he is free, then there is a womant to whom he has not proposed yet. Also, that a woman that got engaged can not become free again. Thus, every woman would become engaged at some point; after all proposals have been made, all the women are engaged (since every woman is on every man's list). Since the number of men and women is equal, at that point there are no free man and so algorithm terminates in at most n^2 rounds.

Now, let us argue that the algorithm returns a perfect matching which is, moreover, stable. First, notice that S is always a matching: a pair (m, w) is only added to S if there is no pair containing m (since he must be free to be proposing), and if a pair containing w, if it exists, is removed. Suppose it is not a matching; then at termination there is a free woman. Thus, there is also a free man. The free woman is on his list, so there is still a possible round where he proposes to her. Thus, at the end, every man is paired with a woman (and thus every woman with a man), forming a perfect matching.

But why would this matching be stable? Suppose that the matching returned by the algorithm isn't: that is, there are (m, w) and (m', w') where m and w' prefer each other. Consider the moment when m proposed to w. By then, since he likes w' more, he should

have proposed to w'. So he must have gotten rejected by her, but this is not possible, since if she was with somebody even better than m, she would not have accepted the proposal from m' (women get better and better choices as the algorithm progresses).

Putting it all together, the algorithm terminates and returns a stable matching. □

1.2 Data structures used in the Stable Matching algorithm

We would like to analyse the time complexity of this algorithm. We will be using the standard worst-case asymptotic complexity notation. Recall that $f(n) \in O(g(n))$ means that $\exists c > 0, n_0$ such that $\forall n > n_0, f(n) \leq cg(n)$. That is, if $f(n)$ is an upper bound on the number of unit steps the algorithm takes on an input of length n, for any n, then if $f(n) \in O(g(n))$, then we say that the asymptotic running time of the algorithm in question is $O(g(n))$. Most of the algorithms we will see have running time $O(n)$ (linear), $O(n \log n)$, $O(n^2)$ (quadratic); if the algorithm has running time $O(n^c)$ for some constant c it is said to run in polynomial time (which is considered to be efficient).

We have shown above that Gale-Shapley algorithm terminates in n^2 steps. Thus, if each round can be implemented in constant time, this algorithm is very efficient: it's running time is $O(n^2)$. Note that the size of the input is $\log n + 2n^2 \log n$ bits, or $2n^2$ words (where every number is considered to be stored as one $\log n$-sized word), so in terms of the number of input bits it would be a linear time algorithm. In order to show how to implement each iteration of the "while" loop in constant time, we need to discuss technical details about the data structures used, and do some precomputation. For this algorithm we will only need arrays and linked lists: recall that looking up an array entry, removing the front element from a linked list and adding an element (front or back) to a linked list are all constant time operations.

We are given, as an input, the preference arrays W and M, where $W[w_i, j] = m$ if m is j^{th} in w_i's preference list, and $M[m_i, j] = w$ if w is j^{th} in m_i's preference list, respectively. Additionally, we need to precompute the following data structures.

- Linked list $Free$ of free men (starts containing all numbers from 1 to n)
- Array $Next$, where $Next[m] = i$ if $w = M[m, i]$ is the next woman m would propose to.
- Array $Current$, where $Current[w] = m$ if m is the man w is currently matched with, and a dummy null value otherwise. This array will essentially store S.
- Array $Ranking$, where $Ranking[w, m] = i$ if m is i^{th} in w's preference list (to speed up comparison of m_i with $m_k = Current[w]$).

Each of these arrays can be precomputed by one pass over the input matrices, therefore the time complexity of the preprocessing step is no more than the time complexity of the actual algorithm. Thus, the sum of the preprocessing time and the algorithm run time is still $O(n^2)$. Also, note that it is enough to maintain $Current$ to have the information about the matching, so there is no need to keep a separate S. And using these data structures, each operation inside the loop can be done in constant time: checking if there is a free man m_i, taking m_i off the list of free men, finding $w_j = Next[m]$ to propose. Checking if w_j is engaged is done by checking if $Current[w_j] = null$, and choosing between m_i and m_k by comparing $Ranking[w_i, Current[w_j]]$ and $Ranking[w_j, m_i]$. Finally, updating $Next[m_i]$ and $Current[w_j]$, and adding either m_i or m_k to the $Free$ list takes constant time as well. Therefore, one iteration of the "while" loop takes constant time, making the algorithm run time $O(n^2)$.

1.3 Properties of the matching returned by the algorithm

There are two (related) facts – first, that the algorithm always+ returns the same matching no matter in what order men propose. Second, men get their best possible choice ("possible" as in there is a stable matching in which this is their choice: the KT book calls it "valid partner"). Moreover, women get their worst valid partner.

In order to show that the algorithm always returns the same matching independently of the order of proposals, just need to show that it returns the (unique) matching in which every man gets his best valid partner.

Here, it is not even obvious that such a matching exists: why can't two men have the same best valid partner? And how is it possible that every man gets the best choice?

Lemma 1 *The Gale-Shapley algorithm always returns the unique matching in which every man is paired with his best valid partner.*

Please see KT book for the proof.