CS 6783 (Applied Algorithms) – Lecture 1

Antonina Kolokolova

January 6, 2012

In this course we will review, and study more deeply the main algorithm design paradigms and techniques. In particular, we will look at greedy algorithms, dynamic programming, backtracking; we will also consider such famous techniques as Fast Fourier Transform (FFT) and Integer Linear Programming. The main emphasis will be on applications. There will be no programming requirements for this course, all algorithm design will be done in pseudocode.

We will start with a clean, yet quite applied problem of Stable Marriage.

1 Stable matching problem

Imagine a situation where there are two types of entities (say organizations and applicants, for example interns applying to companies for their internships or students applying to supervisors for graduate studies). Each of them has a ranking on entities of the opposite type: for example, a student would have a ranking of supervisors with whom he would prefer more to work, and each supervisor, in turn, would have a ranking of students that applied to her.) Now, the question is to find the "best" way of matching the two: matching supervisors with students, companies with interns and so on.

First, let's simplify the problem. Suppose there is exactly the same number of both types, and each of them has to be matched with exactly one of the other type. The classical toy problem description talks about n men and n women, each intending to marry one person of the opposite gender (hence the "stable marriage problem" name). In graph representation, we have a bipartite graph with n vertices on both sides, and looking for a perfect matching in it.

But finding a perfect matching is not quite enough for this problem, because we also want to take into account the rankings. Also, we have not yet defined what it means for the matching to be "good". Would it mean that every applicant (man) gets his best choice? Or would best

be the best for the other side? When economists Gale and Shapley considered this problem, their main concern was not making everybody happy, but, rather, stability: it would have to be a matching such that nobody would like to change their matching to another person, and have that other person accept the switch. That is, the instability is when there are, say, two applicants a_1 and a_2, and two supervisors s_1 and s_2 such that a_1 got assigned to s_1 and a_2 to s_2, but really both a_2 would rather be matched with s_1, and s_1 would prefer a_2 over a_1. In the language of "stable marriage", this is described as a a married man and a married woman who both prefer each other to their spouses – not a stable situation.

Example 1 Suppose there are two supervisors s_1 and s_2, and two applicants a_1 and a_2. Consider the following possibilities. One is when both a_1 and a_2 would prefer s_1, and both s_1 and s_2 would prefer a_1. In that case, the applicant everybody likes gets matched with the supervisor everybody likes; neither of them has an incentive to switch to the other choice, so it is stable. If, on the other hand, a_2 were matched to s_1 and a_1 to s_2, it would not be stable since a_1 and s_1 prefer each other to their pairs.

Now suppose a_1 prefers s_1, a_2 prefers s_2, but s_2 would rather be matched with a_1, and s_1 prefers a_2. In this case, there are two possible matchings that are both stable: either applicants get their choice (and then they have no desire to switch), or the supervisors are, similarly, happier with their current match.

Notice that in the last example there were two possible matchings, both of which were stable. This leads to a question:is it always the case that there would be a stable matching? We will show there is always one by producing an algorithm that is guaranteed to find such matching for any rankings. But the existence of such a matching is far from a trivial problem: a slight modification of this problem called "stable roommates" problem, in which the graph is not bipartite (there is a group of $2n$ people, and each of them can be paired with any of the other $2n-1$) has instances for which no stable matching exists.

1.1 Solving the stable marriage problem

Here is the main idea of the algorithm due to Gale and Shapley that solves the stable matching problem. Imagine the following scenario. Suppose the applicants ("men" in the marriage version terminology) start by contacting ("proposing to") the supervisors ("women"). They only contact a supervisor if they are not currently assigned to any. When such an applicant contacts a supervisor, there could be three possibilities. Either the supervisor is free and looking for applicants, or she is already matched with another applicant. In the first case, she says "maybe" to the applicant, and they are considered matched, for now. If she is already matched with somebody, though, there are two possibilities: either she likes her current match more (in which case she says "no" and the applicant goes on to propose to another supervisor), or she says "sorry, I found somebody better" to her current match, and takes the new applicant. For example, suppose a_1 is applying to s_2, and s_2 is already matched with a_2, although s_2 prefers a_1. Then the match (a_1, s_2) is created and a_2 becomes

free. It makes sense for the applicant to start with the person whom they like the most. It also makes sense that once they got a "no" from somebody, there is no point in trying them again. We will talk more about this when discussing the correctness of the algorithm.

The input to the algorithm will be the two sides of the bipartite graph, let's call them applicants $a_1, \ldots, a_n$ and supervisors $s_1, \ldots, s_n$, together with $2n$ preference lists, each of length n. We assume that for every applicant a_i, for any to supervisors s_j and s_k, both s_j and s_k are in a_i's list, and either s_j comes before s_k, or s_k comes before s_j, but they are not equal. The output of the algorithm will be the list of matched pairs.

StableMarriage(n, A, S)

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

7 **return** M