University of Washington
Department of Computer Science and Engineering
CSE 417, Autumn 2020

September 29, 2020

Homework 1, Due Wednesday, October 7, 2020

Turnin instructions: Electronic submission on gradescope using the CSE 417 gradescope site. Submit the assignment as a PDF, with separate pages for different numbered problems. Problems consisting of multiple parts (e.g., 2a, 2b) can be submitted on the same page.

Problem 1 (10 points):

Let $I = (M, W)$ be an instance of the stable matching problem. Suppose that the preference lists of all $m \in M$ are identical, so without loss of generality, m_i has the preference list $[w_1, w_2, \ldots, w_n]$. Show that there is a unique solution to this instance.

Problem 2 (10 points):

Show that the stable matching problem may have an exponential number of solutions. To be specific, show that for every n, there is an instance of stable matching on sets M and W with $|M| = |W| = n$ where there are at least c^n stable matchings, for some $c > 1$. (Hint: Suppose you have an instance of size n with k solutions, show that you can create an instance of size $2n$ with k^2 solutions.)

Problem 3 (10 points):

(Adapted from text, page 28, exercise 8.) For this problem, we explore the issue of *truthfulness* in the Gale-Shapley algorithm for Stable Matching. Show that a participant can improve its outcome by lying about its preferences. Consider $w \in W$. Suppose w prefers m to m', but m and m' are low on w's preference list. Show that it is possible that by switching the order of m and m' on w's preference list, w achieves a better outcome, e.g., is matched with an m'' higher on the preference list than the one if the actual order was used.

Programming Problem 4 (10 points):

Implement the stable matching algorithm.

You are free to write in any programming language you like (but Java is recommended). The quality of your algorithm may be graded (but you can use the one in the book), but the actual quality of the code will not be graded. The expectation is that you write the algorithmic code yourself - but you can use other code or libraries for supporting operations. You may use a library to generate random permutations (although this can be done as a four-line algorithm.) Submit your code as a PDF.

Make sure that you test your algorithm on small instance sizes, where you are able to check results by hand. A collection of sample instances are provided.

Run your algorithm on the following instance of size $n = 4$. (You can just hard code this as an input into your program.) The preferences for M's are given by the following matrix (where the i-th row in the ordered list of preferences for m_i.

$$\begin{bmatrix} 2 & 1 & 3 & 0 \\ 0 & 1 & 3 & 2 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

and the preferences for the W's are given by the matrix:

$$\begin{bmatrix} 0 & 2 & 1 & 3 \\ 2 & 0 & 3 & 1 \\ 3 & 2 & 1 & 0 \\ 2 & 3 & 1 & 0 \end{bmatrix}$$

Give the resulting matching that is found, along with the list of proposals performed by the algorithm.

Programming Problem 5 (10 points) :

Write an input generator which creates completely random preference lists, so that each M has a random permutation of the W's for preference, and vice-versa. The purpose of this problem is to explore how "good" the algorithm is with respect to M and W. (There is an interesting meta-point relating to algorithm fairness that can be made with this problem.)

We define "goodness" of a match as the position in the preference list. We will number positions from one (not zero as is standard for array indexing.) Note that lower numbers are good. To be precise, suppose m is matched with w. The $mRank$ of m (written $mRank(m)$) is the position of w in m's preference list, and the $wRank$ of w is the position of m in w's preference list. We define the $MRank$ of a matching to be the sum of all of the $mRank(m)$ and the $WRank$ of w to be the sum of all of the $wRank(w)$. If there are n M's (and n W's), we define the $MGoodness$ to be $MRank/n$ and the $WGoodness$ to be $WRank/n$.

As the size of the problem increases - how does the goodness change for M and W? Submit a short write up about how the goodness varies with the input size based on your experiments. Is the result better for the M's or W's? You will probably need to run your algorithm on inputs with n at least 1,000 to get interesting results.