CS 331 Assignment #6

Greg Plaxton

April 16, 2014

NOTE ADDED 4/18/14. There is an error in the original version of the assignment related to tie-breaking issues when there are items of equal quality. For the exercises in Section 1.1, we have repaired the error by choosing an MWMCM M that satisfies a certain sortedness property. (In the numbered list of four values that we fix at the start of Section 1.1, we now fix M last because the sortedness property is defined in terms of β.) This property is needed for solving Exercise 1. The error also has an impact on the programming part of the assignment. This is discussed in detail in a note that has been added at the end of Section 1.2; the main point is that when sorting the items by quality, ties should be broken using the item indices. END OF NOTE.

There are three parts to this assignment. The first part, described in Section 1.1 below, consists of four paper-and-pencil exercises. You are required to turn in solutions to any two of these four exercises. (If you turn in more than two solutions, the grader will arbitrarily choose two of them to grade.) You are encouraged to attend the discussion sections and office hours to get hints on how to solve these exercises. You are also permitted to ask for clarifications on Piazza. You are not allowed to work with other students on these exercises. This part is due *at the start of class* on Wednesday, April 23. Please note that no extension will be given on this part of the assignment!

The second part, described in Section 1.2 below, consists of a programming task that is due by 8pm on Monday, April 28. As in preceding assignments, you should follow the instructions in the document entitled "Guidelines for Programming Tasks", which may be found in the *Assignments* section of the class website. For this programming assignment, the secondary deadline is 8pm on Monday, May 5.

The third part, described in Section 2 below, consists of recommended exercises. There is nothing to be turned in for this part of the assignment, as these exercises will not be graded. You are encouraged to work on the recommended exercises in order to prepare for the next test.

1 Programming & Problem Solving

In Assignments 4 and 5, we developed a theoretical framework for pricing the items in an arbitrary unit-demand auction. In Assignment 5, you wrote a program to implement this

framework for the special class of unit-demand auctions that we had previously studied in Assignments 1 through 3. In this assignment, we develop and implement a faster algorithm for the same task.

1.1 Exercises

In order to simplify our notation, throughout this section we fix the following values.

1. Fix a configuration $G = (U, V, E)$ as defined in Assignment 4. (So G has two dummy bidders for each item, one dummy bidder for the reserve price, and one dummy bidder for the start price.)

2. Fix integer n equal to $|V|$.

3. Fix a bijection β from $[n]$ to V such that the quality of item $\beta(i-1)$ is at most that of item $\beta(i)$ for $0 < i < n$. Note: For any nonnegative integer k, we use the expression $[k]$ to denote the set $\{0, \ldots, k-1\}$.

4. Fix an MWMCM M of G such that the following condition holds for all integers i and j such that $0 \leq i < j < n$: if the bid u that is matched in M to item $\beta(i)$ is linear, and the bid u' that is matched in M to item $\beta(j)$ is linear, then the slope of bid u is at most the slope of bid u'. Remark: Such an MWMCM is guaranteed to exist by Exercise 3 of Assignment 2.

We now introduce a number of additional definitions.

1. Let α denote the function from $[n]$ to U such that, for all i in $[n]$, the bid $\alpha(i)$ is matched in M to item $\beta(i)$.

2. For any integer i in $[n]$, let $\sigma(i)$ denote the maximum integer j in $[i]$ such that bid $\alpha(j)$ is linear (i.e., $\alpha(j)$ is not a single-item bid); if no such integer j exists, we define $\sigma(i)$ to be -1.

3. For any integer i in $[n]$, let $\tau(i)$ denote the minimum integer j such that $i < j < n$ and $\alpha(j)$ is linear; if no such integer j exists, we define $\tau(j)$ to be n.

4. For any price vector p for G, and any integers i and j in $[n]$, we define $f(p, i, j)$ as $p_{\beta(j)} - p_{\beta(i)}$.

5. For any integers i and j in $[n]$, we define $g(i, j)$ as the quality of item $\beta(j)$ minus the quality of item $\beta(i)$.

6. For any integer i in $[n]$ such that bid $\alpha(i)$ is linear, let $s(i)$ denote the slope of $\alpha(i)$.

7. For any price vector p for G, any integer i such that $-1 \leq i \leq n$, and any integer j in $[n]$, we define $gap(p, i, j)$ as follows. If i belongs to $\{-1, n\}$ or bid $\alpha(i)$ is a single-item bid, then we define $gap(p, i, j)$ to be zero. Otherwise, we define $gap(p, i, j)$ as

$$\max(0, s(i) \cdot g(i, j) - f(p, i, j))$$

   The following facts are easy to prove: (1) $gap(p, i, j) > 0$ implies that (M, p) has a violation of Stability Condition 2 that can be eliminated by raising the price of item $\beta(j)$ by at least $gap(p, i, j)$ (put differently, a possible next "move" for the pricing algorithm that you implemented in Assignment 5 is to increase the price of item $\beta(j)$ by $gap(p, i, j)$); (2) If $gap(p, i, j) = 0$ for all integers i and j in $[n]$, then (M, p) satisfies Stability Condition 2.

8. For any price vector p for G, any integer i such that $-1 \leq i \leq n$, and any integer j in $[n]$, we define $update(p, i, j)$ as the price vector p' that is equal to p except that $p'_{\beta(j)}$ is equal to $p_{\beta(j)} + gap(p, i, j)$.

9. For any price vector p for G, and any integer i in $[n+1]$, we define the predicate $L(p, i)$ as "$gap(p, \sigma(j), j) = 0$ for all integers j such that $0 \leq j < i$".

10. For any price vector p for G, and any integer i in $[n+1]$, we define the predicate $R(p, i)$ as "$gap(p, \tau(j), j) = 0$ for all integers j such that $i \leq j < n$".

Exercise 1. Let integers i, j, and k be distinct integers in $[n]$ such that j lies between i and k, i.e., either $i < j < k$ or $k < j < i$. Assume that bids $\alpha(i)$ and $\alpha(j)$ are linear. Prove that if $gap(p, i, j) = 0$ and $gap(p, j, k) = 0$, then $gap(p, i, k) = 0$.

Exercise 2. Let p be a price vector for G such that $L(p, n) \wedge R(p, 0)$ holds. Prove that $gap(p, i, j) = 0$ for all integers i and j in $[n]$. Hint: Make use of Exercise 1.

The proof of the lemma below follows straightforwardly from the definitions.

Lemma 1. *Let p be a price vector for G and let i be an integer in $[n]$ such that $L(p, i)$ holds. Let p' denote the price vector $update(p, \sigma(i), i)$. Then $L(p', i + 1)$ holds.*

Exercise 3. Let p be a price vector for G such that $L(p, n)$ holds. Let i be an integer such that $0 < i \leq n$ and $R(p, i)$ holds. Let p' denote the price vector $update(p, \tau(i-1), i-1)$. Prove that $L(p', n) \wedge R(p', i - 1)$ holds.

Exercise 4. Let p be a price vector for G such that the predicate $P(G, M, p)$ (defined in Assignment 5) holds. Further assume that (M, p) satisfies Stability Condition 3. Describe an $O(n)$-time algorithm to compute the minimum stable price vector for G. You may assume that the bijection β has already been determined and is provided in an array. Hint: Make use of Exercise 2, Lemma 1, and Exercise 3.

1.2 Programming Task

The programming task associated with this assignment is the same as in Assignment 5. The only difference is that in this assignment you are required to process each successive bid insertion in $O(n)$ time. Below we sketch how to do this.

Your program will use the same algorithmic framework as in Assignment 5. As such, when a new bid is added, your program will iteratively perform price increases in order to eliminate all violations of Stability Conditions 2 and 3. Within this framework, the precise number of iterations performed can vary greatly depending on the order in which the violations are addressed.

Remark: This situation is reminiscent of the Ford-Fulkerson algorithm for the network flow problem. Recall that the Ford-Fulkerson algorithm — which performs an arbitrary augmentation at each iteration — does not have worst-case polynomial time complexity, while the Edmonds-Karp algorithm — the special case of the Ford-Fulkerson algorithm in which we always augment along a shortest path from the source to the sink in the residual graph — does have worst-case polynomial time complexity. End of remark.

Each time a new bid is inserted, your program will eliminate all violations of Stability Conditions 2 and 3 in two phases as follows.

In the first phase, your program will eliminate all violations of Stability Condition 3. In order to accomplish this in $O(n)$ time, you will exploit the fact that the sequence of MWMCMs provided to you in the input (or the sequence that you choose to compute on your own) is computed using the algorithm of Assignment 3. As such, each time a new bid is added, exactly one bid (either the new bid, or one of the n bids that was matched just prior to the insertion of the new bid) is added to the set of unmatched bids, and no bid ever transitions from unmatched to matched. It is easy to identify the newly unmatched bid in $O(n)$ time. Observe that all violations of Stability Condition 3 involve this newly unmatched bid. Hence it is easy to eliminate all violations of Stability Condition 3 in $O(n)$ time.

In the second phase, your program will eliminate all violations of Stability Condition 2 using the approach of Exercise 4.

NOTE ADDED 4/18/14. The purpose of this note is to explain how to ensure that your program works correctly when some items are of equal quality. Observe that, in order to employ the approach of Exercise 4 as recommended above, you need to compute a suitable function β with the properties stated in Section 1.1. In effect, this means that you need to sort the items in nondecreasing order of quality. If there are equal-quality items, then there is more than one function β with the required properties. Recall that in the MWMCM algorithm of Assignment 3, we also sort the items in nondecreasing order of quality. In order for the approach of Exercise 4 to work correctly in all cases, the ordering of the items used to compute the prices (i.e., the function β) should be the same as the ordering of the items used to compute the MWMCM M in the algorithm of Assignment 3. (Doing so ensures that the sortedness property associated with the definition of M in Section 1.1 is satisfied.) If you are ignoring the MWMCMs provided in the input (i.e., you are using the algorithm of Assignment 3 to compute an MWMCM of your own), then you should just sort the items

one time, and use the same sorted ordering for both the MWMCM computation and the price computation. However, if you are using the MWMCMs that are provided in the input, then when you sort the items in nondecreasing order of quality (to determine β), you need to break any ties using the item indices; so, for example, if items 2 and 7 each have quality 10, then item 2 should come before item 7 in the sorted ordering. The reason is that the latter tie-breaking convention will be used to generate the sequence of MWMCMs that are provided in the input. END OF NOTE.

2 Recommended Exercises

1. Prove that the number of *finite* subsets of $\{0,1\}^*$ is countably infinite.

2. Let S denote the set of all finite strings over the alphabet $\{a\}$, i.e., $S = a^* = \{\epsilon, a, aa, aaa, \ldots\}$. Prove that there is an undecidable language over the alphabet $\{a\}$, i.e., an undecidable subset of S.

3. Problem 11.5, page 653.

4. Problem 11.8, page 655.

5. Problem 13.9, page 788.