Assignment #6

Greg Plaxton

November 19, 2012

There are two parts to this assignment. The first part is described in Section 1 below, and may be done with a partner. The proofs associated with the first part are due at the beginning of class on Wednesday, November 28.

The second part, described in Section 2 below, consists of a number of exercises related to the lecture material, and is due at the beginning of class on Wednesday, December 5. Each student should work on this part separately.

NOTE: Here is a summary of the changes made to this document since it was handed out in class.

1. In Section 1, the original document referred to a "Lemma 7", but there was no Lemma 7. Lemma 8 has now been renamed to Lemma 7; the bonus problem is to prove this lemma. The basic task is to prove four of Lemmas 1 through 6.

2. In Exercise 4(b), the case conditions mentioned in the hints section have been corrected to $0 \leq k \leq m$ and $m < k \leq 2m$.

3. In the statement of Lemma 3 and in the sentence that follows Lemma 3, I had mistakenly written "Algorithm 2" instead of "Algorithm C".

1 Programming & Problem Solving: Mooov Around

In the previous assignment you implemented an algorithm called Algorithm C for the Mooov Around problem with general (i.e., not necessarily strict) preferences. In the sequence of lemmas stated below, we establish a number of key properties of Algorithm C. Your task is to prove any four of Lemmas 1 through 6. You have the option to also turn in a proof of Lemma 7 for bonus credit. In proving a given lemma, you should feel free to make use of any of the facts or lemmas stated before it. Fact 1 below is straightforward to prove.

Fact 1: Algorithm C performs $O(n^2)$ revelation actions.

Lemma 1: If Algorithm C performs an action that maps the current configuration G to a new configuration G', then $\mathit{satisfied}(G) \subseteq \mathit{satisfied}(G')$.

Lemma 2: If G is a configuration such that a trading action is enabled for cycle C, then at least one student on C belongs to $\mathit{unsatisfied}(G)$.

Lemma 3: Algorithm C performs $O(n)$ trading actions.

Fact 1 and Lemmas 1 through 3 imply that Algorithm C terminates after performing $O(n^2)$ actions.

Lemma 4: If G is a configuration such that *unsatisfied*(G) is nonempty, then some action is enabled in G.

The preceding lemma implies that Algorithm C terminates in a configuration G such that *unsatisfied*(G) is empty. The next result is somewhat more difficult to prove. We state it as a fact because we do not ask you to prove it.

Fact 2: Assume that Algorithm C performs an action mapping the current configuration G to a new configuration G'. Then for any space j, we have *distance*$(G, j) \leq$ *distance*(G', j).

Lemma 5: Assume that Algorithm C performs an action mapping the current configuration G to a new configuration G'. Let student i and space j be such that *next*$(G, i) = j$. Prove that if *distance*$(G', j) =$ *distance*(G, j) then *next*$(G', i) =$ *next*(G, i).

For any configuration G, we define *frozen*(G) as the set of all students i such that *distance*$(G,$ *space*$(G, i)) = \infty$.

Lemma 6: Assume that Algorithm C performs an action mapping the current configuration G to a new configuration G'. Let i be a student in *frozen*(G). Then i belongs to *satisfied*(G), *space*$(G', i) =$ *space*(G, i), and i belongs to *frozen*(G').

In Assignment 4 we proved that the output allocation of Algorithm B is uniquely determined, i.e., it does not depend on how the nondeterminism is resolved. Lemmas 5 and 6 are useful for establishing the following result, which is nontrivial to prove. We state it as a fact because we do not ask you to prove it.

Fact 3: The output allocation of Algorithm C is uniquely determined.

In Assignment 4 we proved that Algorithm B is strategyproof. Using a similar approach, we can establish the following result. We state it as a fact because we do not ask you to prove it.

Fact 4: Algorithm C is strategyproof.

Lemma 7: The allocation produced by Algorithm C is strictly Pareto-efficient. Hints: Let $G_0, G_1, \ldots, G_k$ denote the sequence of configurations associated with an execution of Algorithm C. By Lemma 6, we have *frozen*$(G_{\ell-1}) \subseteq$ *frozen*(G_ℓ) for $1 \leq \ell \leq k$. Since every student belongs to *unsatisfied*(G_0), we have *distance*$(G_0, j) = 1$ for every space j, and hence *frozen*(G_0) is empty. Since Algorithm C terminates in a configuration G such that *unsatisfied*(G) is empty, we conclude that every student belongs to *frozen*(G_k). For $1 \leq \ell \leq k$, let U_ℓ denote the set of students *frozen*$(G_\ell) \setminus$ *frozen*$(G_{\ell-1})$. Assume that some allocation π' weakly dominates the allocation π produced by Algorithm C, and derive a contradiction by comparing the sets of spaces allocated to U_1 under π and π', then comparing the sets of spaces allocated to U_2 under π and π', and so on.

It is straightforward to argue that the allocation produced by Algorithm C satisfies individual rationality. Thus Algorithm C provides a strategyproof solution to the Mooov Around problem. It turns out that Algorithm C enjoys other fundamental game-theoretic properties as well. Specifically, the output allocation is in the "core", and is in the "strict core" if that set is nonempty. An allocation π is in the core (resp., strict core) if no subset

of the students can all do better than (resp., can all do at least as well as, with at least one student doing better) in π by trading amongst themselves the set of spaces initially assigned to them by the university. We omit the (nontrivial) proofs that Algorithm C enjoys these core-related properties.

2 Exercises

1. In class we discussed the problem of determining whether a given graph is 3-colorable, and we referred to this decision problem as 3-COL. We also presented a polynomial-time transformation from 3-SAT to 3-COL as part of a proof that 3-COL is NP-complete (see also Section 8.7 of the textbook). The other part of the proof involved arguing that 3-COL belongs to NP, which is straightforward. Given a graph G, we can also ask whether G is 4-colorable; let us call this decision problem 4-COL. Give a polynomial-time transformation from 3-SAT to 4-COL, justifying your answer. (Since it is easy to argue that 4-COL belongs to NP, we conclude that 4-COL is NP-complete.) Hint: Make suitable modifications to the polynomial-time transformation from 3-SAT to 3-COL that we presented in class.

2. Prove that the number of *finite* languages over the alphabet $\{0, 1\}$ is countably infinite.

3. 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.

4. In class we discussed two approximation algorithms for an NP-hard load balancing problem in which we are asked to distribute n tasks, each with a positive integer weight, over m machines in a way that minimizes the maximum load of any machine. The first algorithm that we presented guarantees a maximum load within a factor of two of optimal. The second algorithm that we presented guarantees a maximum load within a factor of $\frac{3}{2}$ of optimal. These two algorithms are also presented in Section 11.1 of the text, where they are referred to as GreedyBalance and SortedBalance, respectively. The purpose of the present question is to develop a tighter analysis of algorithm SortedBalance. Specifically, we will prove that this algorithm achieves an approximation factor of $\frac{4}{3}$. Fix an instance I of the load balancing problem with m machines and n tasks. Number the tasks from 1 to n, and let w_i denote the positive integer weight of task i. Assume without loss of generality that the tasks are numbered in such a way that $w_i \geq w_{i+1}$, $1 \leq i < n$. Let L^* denote the minimum possible maximum machine load. Let k denote the number of tasks with weight greater than $L^*/3$.

   (a) Explain why k is at most $2m$.

(b) Let L^{**} denote the minimum possible maximum machine load when tasks 1 through k are assigned to the machines (and the rest of the tasks are not assigned to any machine). Prove that at the point in its execution when algorithm SortedBalance has processed tasks 1 through k, the maximum load of any machine is exactly L^{**}. Hints: You may find it useful to consider the cases $0 \leq k \leq m$ and $m < k \leq 2m$ separately. In each case, begin by precisely characterizing of the way that algorithm SortedBalance maps tasks 1 through k to the machines.

(c) Prove that algorithm SortedBalance achieves an approximation ratio of $\frac{4}{3}$. Hint: Use the same overall framework as we used to establish the $\frac{3}{2}$ bound, but improve the bound by making use of the results of parts (a) and (b).