# CS369N: Problem Set #2

Due in class on Friday, November 13, 2009

**Instructions:** same as the first homework.

## Problem 6

This problem introduces an algorithm comparison tool that is a weak form of instance optimality, and applies it to online paging algorithms. Fix a set of $N$ pages and a cache size $k$. Let $I_n$ denote the page request sequences of length $n$. For two online algorithms $A$ and $B$, write $A \leq_n B$ if there is a perfect matching $M$ of $I_n$ with itself (i.e., a bijection) such that $\text{cost}(A, z) \leq \text{cost}(B, M(z))$ for every $z \in I_n$. In other words, for every input $z$ to $A$ we can find an equally bad input $M(z)$ of the same length for $B$.

(a) (4 points) Suppose that $A \leq_n B$ for every $n$. Does this have any implications for an average-case analysis of the performance of $A$ and $B$?

(b) (6 points) Recall the algorithms Flush-When-Full (FWF) and Least Recently Used (LRU) from Lecture #2. Prove that it is *not* the case that $FWF \leq_n LRU$ for every $n$.

(c) (10 points) An online paging algorithm is *lazy* if it only evicts a page on a cache miss. Note that FIFO and LRU are lazy, while FWF is not. Prove that for every two lazy online paging algorithms $A$ and $B$, $A \leq_n B$ and $B \leq_n A$ for every $n$.

[This fact explains why it is difficult to separate the performance of different paging algorithms.]

## Problem 7

Recall that in lecture #2 we explored the access graph model of locality of reference (we saw a second model in HW #1, also). This problem proposes an alternative model and explores it with the weak form of instance optimality from the previous problem.

Fix a set of pages $N$ and a cache size $k$. We use a function $f(\cdot)$ to impose locality of reference, as follows: a request sequence is *legal for* $f$ if and only if in every window of $\ell$ consecutive page requests, there are requests for at most $f(\ell)$ *distinct* pages. We obviously assume that $f(\ell)$ is an integer between 1 and $\ell$ (for each $\ell$); we also assume that $f$ is nondecreasing and concave.

(a) (5 points) Prove that unlike the access graph model, this model of locality has no implications for the competitive ratio of an algorithm. Precisely, prove that for every algorithm $A$ and function $f$ with $f(2) \geq 2$ and $\lim_{\ell \to \infty} f(\ell) = |N|$, the competitive ratio of $A$ on sequences legal for $f$ is the same as its competitive ratio for arbitrary sequences.

(b) (10 points) For algorithms $A$ and $B$ and a function $f$, define $A \leq_n^f B$ as in Problem 6 except with $I_n$ replaced by the subset of length-$n$ sequences that are legal for $f$. Exhibit a function $f$ that separates the FIFO and LRU algorithms in the sense that: (i) $LRU \leq_n^f FIFO$ for every $n$; and (ii) it is not the case that $FIFO \leq_n^f LRU$ for every $n$.

## Problem 8

This problem studies a way of combining algorithms that are optimal in different senses (e.g., one optimized for the worst case and another optimized for the average case). As a concrete application, suppose a sequence of jobs arrive one-by-one, online. When a job $j$ shows up, we learn its processing time $p_{ij}$ on each of $m$ machines $i$. The goal is to schedule all of the jobs on the machines to minimize the makespan (the biggest load on a machine, where the load on a machine is the sum of processing times of jobs assigned to it).

The goal is to get the best of both worlds of two different online algorithms for the problem, $A$ and $B$. The motivation is that $A$ is an online algorithm with good worst-case competitive ratio (like $O(\log n)$) but no knowledge about the input; while the algorithm $B$ has a guess as to what the jobs and their processing times will be and, if the guess is correct, will produce an accordingly excellent schedule (say within $O(1)$ of optimal). But $B$ may have terrible performance if its guess is wrong.

(a) (10 points) Given two online algorithms $A$ and $B$ as above (deterministic, say), show how to combine them into a single "master algorithm" $C$ with the following property: for every input $z$,

$$\text{cost}(C, z) \leq 2 \cdot \min\{\text{cost}(A, z), \text{cost}(B, z)\}.$$

[Hint: imagine running $A$ and $B$ in parallel, and consider very simple rules to decide which one $C$ should pay attention to.]

(b) (5 points) Show by counterexample that the constant 2 in (a) cannot be improved in general for this problem.

## Problem 9

(15 points) Recall that in the *Vertex Cover* problem, you are given an undirected graph $G = (V, E)$ where each vertex has a nonnegative weight $w_v$. The goal is to compute the subset $S$ of $V$ of minimum total weight with the property that every edge has at least one of its endpoints in $S$.

Call a Vertex Cover instance $\gamma$-*stable* if its optimal solution $S^*$ remains optimal even after each vertex $v$ is scaled by an arbitrary factor $\sigma_v \in [1, \gamma]$. Prove that in $\Delta$-stable Vertex Cover instances, the optimal solution can be recovered in polynomial time. (Here $\Delta$ denotes the maximum degree of the graph.)

## Problem 10

(15 points) This problem considers a planted model for graph coloring, to complement the ones we saw in lecture for the minimum bisection and maximum clique problems. Fix an integer $k$ (which you should view as a constant), a number $p \in (0, 1)$ (also constant), and an integer $n$ (which you should think of as going to infinity). Consider generating a random $k$-colorable graph as follows:

1. Each vertex is independently given a label uniformly at random from $\{1, 2, \ldots, k\}$.
2. For each pair of vertices with endpoints with different labels, include the corresponding edge (independently) with probability $p$.

The ensuing graph is $k$-colorable with probability 1, with the color classes corresponding to the subsets of same-labeled vertices.

Design a polynomial-time algorithm that recovers the planted $k$-coloring in such a random graph with high probability (for large $n$).

[Hint: You might want to review Lectures #3 and #4 for some algorithmic and analytical ideas that could be useful. For example, you might want to think about common neighbors.]