CS 787: Advanced Algorithms	**Homework 3**
Out: 11/14/07	**Due: 12/07/07**

Guidelines

Same as always.

Exercises

1. Prove that FIFO is k-competitive for caching.
2. In class we proved that Move-To-Front is 2-competitive for the list update problem by claiming that at any step, the increase in our potential function is no more than $2p' - p + 1$ plus the number of paid exchanges made by OPT, where p is the position of the currently accessed element in our list and p' is the position of that element in the optimal list. In class we proved this claim assuming that OPT does not move the currently accessed element. Complete the proof of this claim for the case that OPT does move this element to some position ahead of p'.
3. Recall that the Halving algorithm makes at most $\log_2 |C|$ mistakes for learning concept class C. Suppose that we wanted a better bound for learning some functions in C but would settle for a worse mistake bound for learning other functions in C. Can you modify the Halving algorithm to obtain such a guarantee? In particular, suppose that we have a distribution p on functions $f \in C$. Give a modified version for the Halving algorithm that makes at most $\log 1/p_f$ mistakes over a sequence of examples labeled according to $f \in C$. (If a function has a high probability mass under this distribution, we want to make fewer than average mistakes for learning that function.) If f is drawn at random from the distribution p, and then examples are labeled according to it, how many mistakes does your algorithm make in expectation?

Problems

1. Recall that in class we proved that randomized 1-bit LRU is $2H_k$ competitive for the caching problem (k is the size of the cache, and $H_k = \sum_{i \leq k} 1/i$.)
   (a) Prove that when the number of pages N is $k + 1$, randomized 1-bit LRU is H_k-competitive.
   (b) Give an example of a request sequence for $k = 2$ and $N = 4$ where this algorithm is *not* H_k competitive.
2. Recall that in the list update problem, we can swap pairs of neighbors for a cost of \$1 each (these are called "paid exchanges"), but when an item x is accessed, we can also move x up in the list for free. It is tempting to conjecture that the optimal offline algorithm never needs to use paid exchanges. Give an example of a request sequence showing that this is false (i.e., in this example you need to use paid exchanges to be optimal).

   Hint: You can do this with 4 requests to a list of 3 elements.
3. The **online bin-packing** problem is a variant of the knapsack problem. We are given an unlimited number of bins, each of size 1. We get a sequence of iems one by one (each of size at most 1), and are required to place them into bins as we receive them. Our goal is to minimize the number of bins we use, subject to the constraint that no bin to should be filled to more than its capacity. In this question we will consider a simple online algorithm for this problem called **First-Fit** (FF). FF orders the bins arbitrarily, and places each item into the first bin that has enough space to hold the item.
   (a) Prove that FF has competitive ratio 2. (Extra credit: Prove that its competitive ratio is $7/4$.)

(b) Give a sequence of item sizes for which the competitive ratio of FF is no better than $3/2$. Try to get as large a lower bound on the competitive ratio as you can.

4. In the **online call admission** problem, we are given a graph (think of it as a telephony network). We get a request sequence of calls each of which is a path in the graph. Our task is to either admit any given call, and remove the used up edges from the graph (they cannot be reused for any subsequent calls), or reject the call. Our objective is to admit as many calls as we can. We will discuss this problem in line graphs with D edges—these are graphs with $D+1$ nodes $v_0, v_1, \cdots, v_D$, and an edge between v_i and v_{i+1} for every $i \in [0, D-1]$.

   (a) Prove that any deterministic algorithm must have a competitive ratio of $\Omega(D)$ on this graph.

   (b) Assume $D = 2^k$ for some k, and consider the following randomized algorithm. Pick an integer $i \in [1, k-1]$ u.a.r. If a call contains between 2^i and 2^{i+1} edges, and does not overlap with any previously admitted call, then admit the call, otherwise reject it. Prove that this algorithm has a competitive ratio of $O(k) = O(\log D)$. *Hint: What is the c.r. of this algorithm if* **all** *calls have between 2^i and 2^{i+1} edges?*

5. **Tracking a moving target.** Here is a variation on the deterministic Weighted- Majority algorithm, designed to make it more adaptive.

   (a) Each expert begins with weight 1 (as before).

   (b) We predict the result of a weighted-majority vote of the experts (as before).

   (c) If an expert makes a mistake, we penalize it by dividing its weight by 2, but only if its weight was at least $1/4$ of the average weight of experts.

   Prove that in any contiguous block of trials (e.g., the 51st example through the 77th example), the number of mistakes made by the algorithm is at most $O(m + \log n)$, where m is the number of mistakes made by the best expert in that block, and n is the total number of experts.

6. **The "sleeping-experts" problem.** Consider a standard experts setting and suppose that at any time step, only a subset of the experts are available to make a prediction. Can we modify the multiplicative updates algorithm from class to give a low regret bound in this case? More precisely, in this setting, at every time step we get to see which experts are "awake" to make a prediction. We choose one of those experts. Then we get to see the cost vector for that step. The total cost of expert i is the sum of its costs over the steps that i was awake in. Naturally, if an expert is awake for very few steps, we cannot hope to prove that our total cost over all the steps is not much larger than the expert's cost over the steps that it was awake in. So we will aim for a slightly different guarantee.

   Let T_i denote the set of steps when expert i was awake, $\text{cost}_i(\text{ALG})$ denote the expected cost of the algorithm over the steps T_i, and $\text{cost}_i(i)$ denote the cost of expert i over the steps T_i. Then, our goal is to say that $\text{cost}_i(\text{ALG}) \leq (1+\epsilon)\text{cost}_i(i) + O(\frac{1}{\epsilon}\log n)$, and this should hold for every expert i.

   Consider the following variant of the multiplicative updates method for this problem:

   (i) Initialize $w_{i,0} = 1$ for all i.

   (ii) At every step t, let $p_{i,t} = w_{i,t}/W_t$ be the probability of picking an awake expert i, where W_t is the sum of the weights of all the experts awake at that step (and not the total weight of all the experts).

   (iii) Let $c_{i,t}$ denote the cost to expert i at step t. Update weights for awake experts as follows:

$$R_{i,t} = \frac{1}{1+\epsilon}\left(\sum_j p_{j,t}c_{j,t}\right) - c_{i,t}$$

$$w_{i,t+1} = w_{i,t}(1+\epsilon)^{R_{i,t}}$$

   (a) Prove using induction that the total weight of all experts at any step is at most n. (In particular, although individual weights can go up or down, the total weight never goes up).

   (b) Express the weight of expert i at time T in the form of the total expected cost of the algorithm and the total cost of the expert. Use part (a) to prove that $\text{cost}_i(\text{ALG}) \leq (1+\epsilon)\text{cost}_i(i) + O(\frac{1}{\epsilon}\log n)$.