15-750: Algorithms in the Real World **CMU, Spring 2023**
HW #6 (out Tuesday April 11, 2023) **Due:** Sunday April 30, 2023, **no extensions**

Please limit yourself to groups of two (or three) people. We adhere to a "whiteboard" collaboration policy: you should discuss in your group on whiteboards (or equivalents) but then should go away and write down your solutions yourself. You must not share written work. **You must write down the names of the person(s) with whom you collaborate.** In general, you should solve the homework problems using only to material we refer you to (or put on the course page), and not books or other online resources. If you have reason to use any resources we point you to (except the course notes), please cite them.

A word of advice: please try to solve the problem by yourself before working with your group. That is the way to improve your problem-solving skills.

Homeworks will be due at 11:59pm on the due date on `gradescope`. Corrections and changes will appear on the course webpage and on Piazza; please check them regularly.

Problems

1. Here is an online algorithms problem. Tomorrow, you will have 1 hour of server time. Today, a sequence of jobs will arrive at your doorstep. They come with some known runtimes, $r_1, r_2, r_3, \ldots, r_T$ which are real numbers between 0 and 1. As each job comes in, you have to decide to either "keep it" or "pass on it". The total runtime of the jobs you keep cannot exceed 1, and your goal is to get it as close to 1 as possible, so as to not waste your server time. Once you decide to keep a job, you can't later to decide throw it away after all.

   (Incidentally, it doesn't *really* matter whether or not you "know" T, the number of jobs that will pass by today; let's assume you don't necessarily know it.)

   We will analyze potential algorithms in terms of "competitive ratio", but note that this is a *maximization* problem, so the "competitive ratio" on an input sequence $\mathcal{I}$, namely

   $$\mathrm{CR}(\mathcal{I}) = \frac{\text{total runtime your algorithm keeps on input } \mathcal{I}}{\text{largest total runtime a psychic knowing all of } \mathcal{I} \text{ in advance could keep}},$$

   will be a number that is ≤ 1, and you're happy the closer it is to 1.

   (a) Prove that for every number $c > 0$, there does not exist a deterministic algorithm with competitive ratio at least c.

   (b) The above fact is pretty disappointing! Let's help you out a little. Suppose that, at the very beginning of the day, a genie comes to you and grants you *one* yes-or-no question about the incoming sequence $r_1, r_2, \ldots, r_T$. You decide to ask, "Will there be at least one job with runtime at least 1/2?" Show that once you get the truthful answer to that question, you can design an algorithm with competitive ratio at least 1/2.

2. Here is a variation on the deterministic Weighted-Majority algorithm, designed to make it a bit more adaptive.

   - As usual, each expert begins with weight 1.
   - As usual, we predict based on the weighted-majority vote of the experts.

- If an expert makes a mistake, we penalize it by multiplying its weight by 1/2, but *only* if its weight was at least 1/4 of the average weight of experts.

Prove that in any contiguous block of days (e.g., the 51st day through the 77th day), 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 number of experts. Note that in our original Weighted-Majority algorithm, we proved this result only for contiguous blocks that include the first day.

3. Recall the setting of the Hedge (/Multiplicative Weights) problem, where there are N "slot machines" (aka experts), the algorithm plays a probability distribution $p_1^t, \ldots, p_N^t$ on each day, and the Adversary assigns losses $\ell_1^t, \ldots, \ell_N^t$ (between -1 and $+1$) each day. Actually, in class we thought of the algorithm as just outputting the numbers $p_1^t, \ldots, p_N^t$, but in this problem lets use the interpretation that the algorithm actually uses this probability to pick a random slot machine I_t to play on the tth day. Recall from class that we proved the following fact: Given a small parameter $\varepsilon > 0$, the Hedge algorithm's expected total loss is at most

$$\min_{1 \le i \le N} \Big(\sum_{t=1}^{T} \ell_i^t \Big) + \varepsilon T + \frac{\ln N}{\varepsilon}.$$

Let's now consider a new variant (relevant when using Hedge as a meta-algorithm to switch between various algorithms in a system). In this new variant, there is a *switching cost* of K to the algorithm. That is, whenever the algorithm chooses $I_t \neq I_{t-1}$, it experiences an extra loss of K on the tth day. (You can assume there is no switching cost on the 1st day.) So an algorithms expected total loss is

$$\sum_{t=1}^{T} \mathbf{E}\Big[\ell_{I_t}^t + K S_t \Big],$$

where S_t is the random variable which is 1 if $I_t \neq I_{t-1}$, and is 0 if $I_t = I_{t-1}$.

In this problem you will show how to choose I_t so that the new switching costs are not too painful. You will do this in two steps. First, you'll show that the distributions p^{t-1} and p^t that Hedge produces on consecutive days are reasonably "close", for all t. Then you'll show how to sample from these distributions in a way that makes I_t usually equal to I_{t-1}.

(a) Suppose you run the Hedge algorithm (with parameter $\varepsilon \le 1/4$). Show that

$$\|p^t - p^{t+1}\|_1 := \sum_{i=1}^{N} |p_i^t - p_i^{t+1}| \quad \le \quad O(\varepsilon) \sum_i p_j^t |\ell_j^t|.$$

(b) Now let's decide how I_t should be chosen. What we *won't* do is simply pick I_t according to the probability distribution p^t using "fresh random bits". Instead, we will pick it in a way that makes it *correlated* with the previous I_{t-1}. Well... except for on the first round when $t = 1$ and $p_i^1 = 1/N$ for all i; in this round we will simply pick I_1 to be uniformly random from $\{1, 2, \ldots, N\}$.

As for $t > 1$, explain why, after defining $p_1^t, \ldots, p_N^t$, the algorithm can choose some "probability redistribution values" $x_{ij} \ge 0$ such that $x_{ii} = \min(p_i^{t-1}, p_i^t)$ for all i, and $\sum_j x_{ij} = p_i^{t-1}$, and $\sum_i x_{ij} = p_j^t$. (You might like to illustrate your explanation with a small example, say with $N = 3$.)

(c) Having defined the x_{ij}'s, suppose the algorithm actually selects its I_t value by choosing j with probability $x_{I_{t-1}j}/p^{t-1}_{I_{t-1}}$. Show that this indeed causes I_t to equal j with probability p^t_j, as it is supposed to be.

(d) Show that for each t,

$$\mathbf{Pr}[I_t \neq I_{t+1}] \leq \|p^t - p^{t+1}\|_1.$$

(e) Conclude that if $\delta > 0$ is a small given parameter, if we run the above algorithm with $\varepsilon = \delta/K$, the expected total loss is at most

$$\min_{1 \leq i \leq N} \Big(\sum_{t=1}^{T} \ell_i^t \Big) + O(\delta T) + \frac{O(K \ln N)}{\delta}.$$

4. (a) Write code in your favorite programming language that implements the "Zero Sum Game solver" described in Lecture 21. It should take as input: (i) the payoff matrix M; (ii) the Hedge parameter ε (between 0 and 1); (iii) a number of rounds T to play for. Rather than insisting that the payoffs be between -1 and $+1$, you should allow for any payoffs, but you should require that ε be smaller than the largest magnitude payoff.

Given this input, your program should output: (i) the *average* mixed strategy $(\overline{p}_1, \ldots, \overline{p}_{N_1})$ for Alice (this is the average over $t = 1 \ldots T$ of all the vectors $(p^t_1, \ldots, p^t_{N_1})$ Alice played); (ii) the mixed strategy defined empirically by Bob's play (that is, the vector $(\overline{q}_1, \ldots, \overline{q}_{N_2})$ where $\overline{q}_i$ is the fraction of times Bob played action i); (iii) the expected loss in the game when Alice plays with $\overline{p}$ and Bob plays with $\overline{q}$. For this problem, put your source code into your submission document.

(b) The Zero Sum Game of *Underwhelm* is played as follows: On the count of three, Alice and Bob each show some number of fingers between 1 and 5. Say Alice shows a and Bob shows b. If $|a - b| \neq 1$, then whoever played the *smaller* number gets that many dollars from the other person (with no money changing hands if $a = b$). On the other hand, if $|a - b| = 1$, then whoever played the *larger* number gets $a + b$ dollars from the other person.

For example, if Alice plays 3 and Bob plays 5, then Bob pays Alice 3. If Alice plays 4 and Bob plays 1, then Alice pays Bob 1. If Alice plays 3 and Bob plays 4, then Alice pays Bob 7. Note that Underwhelm is a *symmetric* game; Alice and Bob have the same number of actions and the payoffs work the same for both players.

Using your code from part (a) (and trying to set ε very small and T very large), "solve" Underwhelm, coming up with your best approximation to $\overline{p} = \overline{q}$.

(If you have the enthusiasm for it, try to figure out how things change if the players can play any number between 1 and N, where $N > 5$. Trying playing the game yourself in person with your friend, with $N = 5$ or $N = 10^{100}$.)