CIS 320 Problem Set 6

Due: Wednesday 12/08/21

Welcome to Problem Set 6. A few notes before you begin.

- You may collaborate with up to 4 other people. All work must be written up individually. You may collaborate, but you cannot copy. E.g. you can talk to your friends about ideas for a problem together, but you cannot copy their solution and just change some words around. **List your collaborators at the top of your submission**.
- Googling or looking up solutions in any way is not allowed.
- Cheating is not worth it! Your grade in this course does not define your worth as a person, and in 10 years you will not care about your bad grade on a homework assignment. But you will care if you are caught plagiarizing: plagiarizing has serious consequences, including the potential of expulsion.
- These problems are designed to challenge you. Start them early; if we've done our job well writing them, you will have to chew on them for a while before finding the solutions, and that means you will do best if you can sleep on your solutions rather than starting them the night before the due date.
- You do not need to implement anything in code. In fact, please do not. Pseudocode or a clear English explanation of your algorithm are both acceptable: in some cases pseudocode may be clearer than plain English, and in others the plain English might be better.
- If a question asks you to come up with an algorithm with a certain target runtime (say $O(n \log n)$), and you don't know how to achieve this runtime but know how to solve the problem less efficiently (in, say, $\Theta(n^2)$ time), write that down! Depending on how inefficient your solution is compared to the benchmark, it will receive a varying amount of credit.
- For each of our algorithm design questions, you should come up with a *deterministic* algorithm unless the question specifies that a randomized algorithm is wanted.
- All analysis must be mathematically rigorous. Any answer you provide should be proven.
- Remember, we can't evaluate your work if we can't understand it. Communicating mathematical and/or complex ideas is an important skill in computer science; treat your problem sets as practice.
- You should use LaTeX to typeset your solutions.
- Have fun!!

Problem 1. Recall that an equilibrium for a two-player zero-sum game between maximization player Alice and minimization player Bob is a pair of randomized strategies S_A, S_B (i.e. distributions over actions) such that even if Alice knows S_B ahead of time, she has no incentive to deviate from S_A, and similarly, if Bob knows S_A ahead of time, he has no incentive to deviate from S_B.

In lecture we saw that equilibria in zero-sum games can be computed via the polynomial weights algorithm, or by solving linear programs. But for very simple games it is possible to directly compute the equilibrium strategies.

Consider a two-action zero-sum game between Alice and Bob, with the following payoff matrix. Alice's actions correspond to rows of the matrix, and Bob's actions correspond to columns of the matrix:

$$\begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

(a) Assume that there is an equilibrium S_A, S_B of this game so that both players are randomizing — i.e. both Alice and Bob play both of their actions with non-zero probability. Prove that if Bob is playing his equilibrium distribution S_B, then Alice is indifferent between her two actions: the expected payoff for Alice of just playing action 1 is equal to the expected payoff of just playing action 2. Similarly, prove that, given that Alice is playing her equilibrium distribution S_A, Bob is indifferent between his two actions: the expected payoff for Bob of just playing action 1 is equal to the expected payoff of just playing action 2. (5 points)

(b) Given what you proved above, find an equilibrium of the game by solving a system of linear inequalities. (5 points)

Problem 2. You are looking to raise a series A round for your awesome new startup (Uber for cats), and have decided that what matters in tech is not *what* you know, but *who* you know. So you are going to use this principle in seeking out investors. There are n VC partners who invest in your industry (Uber for x), and you can select any k of them to invest in your startup (its that good). You've mapped out exactly which of the partners know which of the others in a giant social network graph you've made with string and pushpins on a corkboard on your garage wall. You want to know if there are k investors S you can select so that each partner i *either* is investing in your startup ($i \in S$) or else *knows* one of the investors $j \in S$. You call this the Connected VC (C-VC) problem.

Prove that the C-VC problem is NP complete. (10 points)

Problem 3. You got your funding! But you're going to need a great app if cats are going to be able to figure out how to use it. Its time to hire engineers — but not just any engineers: you need "10x engineers". Unfortunately these are very hard to hire: they are extremely demanding. You have a collection of n_1 perks you can offer to individual engineers (use of the company Tesla, a rare NFT of an elmo GIF, etc.). But each perk can only be given to one engineer. There are n_2 10x engineers you are considering, and each of them has delivered to you a list of the perks that they require: you can only hire an engineer if you give them *all* the perks they demand. Your problem is to determine if, given your n_1 perks, there is any set of k of the n_2 10x engineers you can hire. You call this the Demanding Software-engineer (D-IS) problem.

Prove that the D-IS problem is NP complete. (10 points)

Problem 4. 10 points Extra Credit! In this problem, we complement the outline of the calibration algorithm from lecture with an important implementation detail: the Learner doesn't actually need to solve a linear program at each round in order to determine her minimax optimal distribution over weather predictions — instead, she can easily find an explicit distribution over her weather predictions that is not necessarily minimax optimal but good enough to guarantee calibration.

Recall that at round s, we have the following bound on the Learner's surrogate loss increase:

$$\Delta_s(p_s, y_s) \leq 2\, V_{s-1}^{p_s m} \cdot (y_s - p_s) + 1.$$

Here, $p_s \in \{1/m, \ldots, m/m\}$ is the (deterministic) weather prediction that the Learner makes, $y_s \in \{0, 1\}$ is the weather chosen by the Adversary, and $\Delta_s(p_s, y_s)$ is the increment in the Learner's surrogate loss in round s resulting from the action pair (p_s, y_s) being played. Furthermore, for each $i \in \{1, \ldots, m\}$, recall that we have defined $V_{s-1}^i = \sum_{t=1}^{s-1} 1[p_t \in B(i)] \cdot (y_t - p_t)$ — which, at the beginning of round s, is just a constant quantity (since it sums up terms over the past rounds $1, \ldots, s-1$).

How should the Learner go about choosing a good p_s (i.e. one that does not increase the surrogate loss by much)? In lecture, we proposed to define a zero-sum game, where the Learner is the minimization player and the Adversary is the maximization player, which we can very explicitly describe as follows:

- the Learner commits to a probability distribution $\hat{p}_s$ over the set of pure strategies $p_s \in \{1/m, \ldots, m/m\}$
- the Learner's pure action (i.e. her actual weather prediction for this round) p_s is sampled from $\hat{p}_s$
- the Adversary responds with a pure strategy $y_s \in \{0, 1\}$
- the resulting cost is $C_s(p_s, y_s) = 2\, V_{s-1}^{p_s m} \cdot (y_s - p_s) + 1$, i.e. the bound on the increase in the surrogate loss from above.

In lecture, we saw via the Minimax theorem that the value of this game was at most

$$B_s := 2 \cdot \frac{T}{m} + 1 = 3.$$

where the last equality follows from the fact that in the end we chose $m = T$. We then observed that if the Learner can guarantee expected cost B_s or lower in all rounds s, then the Learner is calibrated!

As you can remember, in lecture we simply noted that the Learner can achieve the desired bound

$$\max_{y_s \in \{0,1\}} \mathbb{E}_{p_s \sim \hat{p}_s}[C_s(p_s, y_s)] \leq B_s \tag{1}$$

by taking $\hat{p}_s$ to be her *minimax* probability distribution — which we found using a simple linear program.

As it turns out, we can forgo solving the minimax linear program and find an explicit distribution[1] $\hat{p}_s$ that achieves the desired bound (1). The purpose of this problem is to guide you through the process of finding such a distribution. The key to constructing such a probability distribution $\hat{p}_s$ is to analyze the quantities $V_{s-1}^1, \ldots, V_{s-1}^m$ — which, as you recall, we should just think of as fixed given quantities in $[-s+1, s-1]$ which we have no control over.

(a) Suppose $V_{s-1}^m \geq 0$. Show that by deterministically playing $p_s = 1$, the Learner guarantees herself cost at most 1 no matter how the Adversary responds.

(b) Suppose $V_{s-1}^1 \leq 0$. Show that by deterministically playing $p_s = \frac{1}{m}$, the Learner guarantees herself cost at most B_s no matter how the Adversary responds.

(c) Now consider the remaining case when $V_{s-1}^1 > 0 > V_{s-1}^m$.

  (a) Prove that there exists an index $i \in \{1, \ldots, m\}$ such that $V_{s-1}^i \geq 0 \geq V_{s-1}^{i+1}$.

[1] which is potentially not a minimax distribution.

(b) Suppose the Learner decides to play a distribution $\hat{p}_s$ that only randomizes over two actions: $\{\frac{i}{m}, \frac{i+1}{m}\}$. Let q_s be the probability that $\hat{p}_s$ assigns to playing $p_s = \frac{i}{m}$ (so that the probability of playing $p_s = \frac{i+1}{m}$ is $1 - q_s$). Give an expression for $\mathbb{E}_{p_s \sim \hat{p}_s}[C_s(p_s, y_s)]$ in the form $D_1 y_s + D_2$, where D_1, D_2 depend on $q_s, i, m, V_{s-1}^i, V_{s-1}^{i+1}$, but not on y_s.

(c) Given V_{s-1}^i, V_{s-1}^{i+1}, find the formula for such $\hat{q}_s \in [0,1]$ that setting $q_s = \hat{q}_s$ leads to $D_1 = 0$. Conclude that if the Learner plays distribution $\hat{p}_s$ with $q_s = \hat{q}_s$, then no matter what the Adversary plays in response, the Learner guarantees herself cost D_2.

(d) For $q_s = \hat{q}_s$, prove that $D_2 \leq B_s$.

(d) Conclude from the above items that, no matter the values $V_{s-1}^1, \ldots, V_{s-1}^m$, the Learner has an explicit-form distribution $\hat{p}_s$ that (I) guarantees her cost is at most B_s, no matter the Adversary's response; and (II) randomizes over at most two predictions in $\{\frac{1}{m}, \ldots, \frac{m}{m}\}$.