This HW may be solved collaboratively. Remember to justify the correctness of your algorithms. All problems are worth 33\frac{1}{3} points.

Exercises (just for practice, not for submission)

1. Optimizing Production. Your company is building new economic stoves that reduce the amount of greenhouse gas emissions. You have $k$ different kinds of stoves you can build at your factory. There are $L$ different basic components that are assembled together to build these stoves, and these are numbered $1, 2, \ldots, L$. For each $i \in \{1, 2, \ldots, k\}$ and $j \in \{1, 2, \ldots, L\}$, stove type $i$ uses up $A_{ij} \geq 0$ amounts of component $j$. Moreover it requires $t_i$ person-hours to assemble, and you have a total of $T$ person-hours available each year.

There are other contractual obligations: (a) you must build at least $d_i$ units of stove $i$, (b) the number of stoves of type 1 and 2 built can differ by at most 2\% of the total number of stoves built by your company, and (c) if your company uses more than $M$ units of component 1 then it must install an air filter which costs $F$.

Finally, the cost per unit of each component $j$ is $c_j$, and the assembly cost for each stove of type $i$ is $\lambda_i$. (This need not be related to the number of person hours $t_i$.) The sale price of stove $i$ is $S_i$. Give a solution using LPs (more than one may be needed) that computes the number of stoves of each type needed to maximize profit. (It's fine to have a fractional solution for this problem.)

2. Consider the Max-Coverage problem, which is like Set Cover, but where we can only pick $K$ sets, and we want to maximize the number of covered elements. If the optimal solution covers $C$ elements, show that the greedy algorithm covers at least $(1 - 1/e) \cdot C$ elements. (Hint: mimic the set-cover analysis.)

3. For the optimal search problem, show that flipping a coin and starting the search randomly going left or right yields a competitive ratio of 7.

4. Show instances of list update on which the frequency count approach has non-constant competitive ratio.

Problems

1. (The Project Manager’s Problem.) You have $m$ projects $p_1, p_2, \ldots, p_m$ you would like to complete before the end of the semester. It works like this: there are $n$ basic tasks, where task $j$ costs $c_j$ to complete. For each project $p_j$, there is a subset $P_j \subseteq \{1, \ldots, n\}$ of these basic tasks. If you complete all these tasks in $P_j$, the project $p_j$ will be completed, giving you $V_j$ dollars of value. You are given as input two integers $C$ and $K$, and you want to find out if there is a subset of basic tasks you can finish whose total cost is at most $C$, such that doing those tasks will have completed projects whose total value is at least $K$.

For example, let $n = 5$ with the basic tasks being $\{a, b, c, d, e\}$. The $m = 6$ projects may be $\{a, b, c\}, \{b, d, e\}, \{c, d\}, \{e\}, \{a, c, d, e\}$, and $\{a, e\}$. If the tasks cost 1 each, and the projects each give value 1, completing the tasks $a, d, e$ costs 3 and gives you value 2 (since you have completed the projects $P_4, P_6$), whereas completing $a, c, d, e$ gives value 4 (since you completed $P_3, P_4, P_5, P_6$).
Prove that this problem is NP-complete, even when the costs \( c_j = 1 \) for all basic tasks \( j \), and values \( V_j = 1 \) for all the project values.

2. **(Order and Method.)** You are given \( n \) jobs, each with some required processing \( p_i > 0 \). You have \( m \) machines on which to schedule these jobs. In lecture you saw a greedy algorithm to schedule jobs in order to approximately minimize the *makespan*, i.e., the maximum completion time over all the jobs.

Here’s a common variant of this problem: there is a partial ordering on these jobs, so that some jobs must be done before others. Imagine a directed graph \( G = (V, E) \) with vertex set \( V = \{1, 2, \ldots, n\} \), and directed edges \( E \subseteq V \times V \). An edge \((i, j)\) implies that in the schedule you find, job \( i \) must have finished when job \( j \) starts. For simplicity, assume that for each edge \((i, j)\) \( \in E \), we have \( i < j \), so that the edges are directed “forwards”. Here’s an example:

Suppose \( n = 5 \) and \( m = 2 \), and \( E = (1, 2), (1, 3), (1, 5), (2, 4), (3, 4) \), and say all \( p_1 = 2, p_2 = 1, p_3 = 3, p_4 = 3, p_5 = 1 \). Then one schedule is to start job 1 at time 0; this runs for the first two timesteps. Then start both 2 and 3 at time 2. Start job 5 when 2 finishes. Finally, start job 4 when 3 finishes.

Give an algorithm that finds a schedule whose makespan is at most two times the optimal makespan. *Hint: Given the graph \( G \) what lower bounds for the makespan can you write down? How would you relate your algorithm’s performance to these lower bounds?*

3. **(Randomized Ski-Rental.)** The eternal dilemma is: buy now, or rent for a little longer? Assume it costs $\( B \) to buy skis, and $1 (per day) to rent. In lecture we saw a deterministic algorithm with competitive ratio \( 2 - 1/B \). Let’s develop an optimal randomized algorithm.

The algorithm’s possible actions are of the form: “rent until day \( w - 1 \); if you still are interested in skiing on day \( w \) then buy the skis”, one such action for every \( w \geq 1 \). And the adversary’s actions are of the form “you will ski for \( t \) days”, one such action for every positive integer \( t \geq 1 \).

For instance if \( w = 4 \) and \( t \leq 3 \) then you pay exactly \( t \), but if \( w = 4, t \geq 4 \) then you buy on day \( w \) and pay \((w - 1) + B\).

(a) Given the pair of actions \((t, w)\), your “pain” \( R_{tw} \) is the ratio between your cost divided by the optimal cost. Let \( R \) be the “pain” matrix, whose rows \( t \) are the adversary’s actions and columns \( w \) are your actions. Write down the expression for \( R_{tw} \).

(b) This matrix has an infinite number of columns and an infinite number of rows. Let’s add one more row, called “row \( \infty \)” for the scenario that we ski forever. Argue that without loss of generality, we can assume the adversary chooses only rows \( t \in \{1, 2, \ldots, B - 1\} \) or \( t = \infty \). Formally, argue that for any \( t \geq B \) we have \( R_{tw} \leq R_{\infty w} \) for all \( w \). This means that any algorithmic strategy against an adversary that chooses only actions from \( \{t < B\} \cup \{t = \infty\} \) is just as good against an adversary that can choose any value of \( t \).

(c) Now that the matrix has just \( B \) rows, argue that any optimal strategy for you can safely put probability 0 on all columns except for \( w \leq B \). Formally, argue that for any \( w > B \) we have \( R_{tw} \geq R_{tB} \) for \( t \in \{1, 2, \ldots, B - 1, \infty\} \).

(d) Write down the \( B \times B \) matrix that results. *Hint: try this for \( B = 2, 3, 4 \) and you will see the pattern.*
(e) Now to design the randomized algorithm. Such an algorithm (randomly) chooses some value of \( w \) and then buys on that day: i.e., it randomly chooses column \( w \) and plays the action corresponding to this column. Suppose column \( w \) is chosen with probability \( p_w \geq 0 \), where \( \sum_{w \leq B} p_w = 1 \) (i.e., you choose exactly one of these actions). Write down an expression for the competitive ratio of the resulting algorithm, in terms of \( p \) and the matrix \( R \)? Now write an LP to compute the values \( p_1, p_2, \ldots, p_B \) that achieve the best competitive ratio.

(f) (2 points) Write a program (in python, say) to solve the LPs for some increasing values of \( B \) and plot the competitive ratios you obtain — observe that the ratios converge to \( \frac{e}{e-1} \) (from below).