GEORGIA TECH S'22 CS 6550/8803: ADVANCED ALGORITHMS & UNCERTAINTY

Homework 2

Out: *Feb 1* Due: *Feb 22*

Instructions:

- Each problem is worth twenty-five points. **Attempt any 4 of the 5 problems.** If you submit all 5 then the TAs might grade arbitrary 4 of them.
- Upload your solutions to the problems as a single PDF on Gradescope. Please anonymize all your submissions (i.e., do not list your name in the PDF), but if you forget, it's OK.
- You may collaborate with any classmates, textbooks, Internet, etc. Please upload a brief "collaboration statement" listing any collaborators as the last page of your non-extra-credit solutions PDF on Gradescope. But after the collaboration, always **write your solutions individually**.
- If you choose to do extra credit, upload your solution to the extra credits as a single separate PDF file to Gradescope. Please again anonymize your submission.
- For each problem, you should aim to keep your writeup below one page. For some problems, this may be infeasible, and for some problems you may write significantly less than a page. This is not a hard constraint, but part of the assignment is figuring out how to easily convince the grader of correctness, and to do so concisely. "One page" is just a guideline: if your solution is longer because you chose to use figures (or large margins, display math, etc.) that's fine.

Problems:

§1 (Scheduling: LP Rounding) Suppose we are given n machines and m jobs where the jth job takes time $p_{ji} \in \mathbb{R}_{\geq 0}$ to be executed on machine i. The goal of this problem is to find an assignment $\sigma : [m] \to [n]$ of all the jobs onto the n machines to minimize the maximum load, i.e., $\min_\sigma \max_{i \in [n]} \sum_{j:\sigma(j)=i} p_{ji}$. Let OPT denote this value for the optimum assignment.

Let $\text{OPT}_{\text{guess}}$ denote the smallest value for which the following LP (where we disallow "large" jobs on a machine) is feasible:

$$\begin{aligned}
&\sum_i x_{ji} \geq 1 && \forall j \in [m] \\
&\sum_j x_{ji} p_{ji} \leq \text{OPT}_{\text{guess}} && \forall i \in [n] \\
&x_{ji} = 0 && \forall j \in [m], i \in [n] \text{ with } p_{ji} > \text{OPT}_{\text{guess}} \\
&x_{ji} \geq 0 && \forall j \in [m], i \in [n]
\end{aligned}$$

We will assume that $\text{OPT}_{\text{guess}}$ is known since this can be found up to a small $1+\epsilon$ factor using binary search. Let x^*_{ji} denote the optimal fractional solution to this LP.

(a) Prove that $\text{OPT}_{\text{guess}} \leq \text{OPT}$.

In the rest of the problem we will use x^*_{ji} to find an integral assignment with maximum machine load $2\text{OPT}_{\text{guess}}$. This will imply a 2-approximation due to (a).

For each machine i, let $w_i = \lceil \sum_j x^*_{ji} \rceil$. Now make a bipartite graph with jobs on the left and machines on the right: make w_i copies of the machine i node on the right, call them $i_1, \ldots, i_{w_i}$ and make a single node on the left for each job j.

For each machine i, sort the jobs in decreasing order of p_{ji}, so that $p_{(1)i} \geq p_{(2)i} \ldots \geq p_{(m)i}$. Place edges from jobs to machine i in the following manner:

- Initialize current-node $c := 1$. Initialize current-job $j := 1$. Initialize job-weight $w := x^*_{j(1)}$. Initialize node-weight-remaining $r := 1$.
- While $(j \leq m)$:
  - (i) If $w \leq r$, add an edge from job (j) to node i_c of weight w. Update $r := r - w$, update $j := j + 1$, $w := x^*_{(j)i}$ (the newly updated j). Keep $c := c$.
  - (ii) Else, add an edge from job (j) to c of weight r. Update $w := w - r$, update $r := 1$, update $c := c + 1$. Keep $j := j$.

In other words, starting from the longest jobs, we put edges totaling weight x^*_{ji} from job j to (possibly multiple) nodes for machine i. We do so in a way such that the longest jobs are on the earliest-indexed copies, and that each machine-copy has total incoming weight at most 1 (actually all but the last copy have incoming weight exactly one, and the last copy has weight at most one).

(b) Given the above bipartite graph, our rounding algorithm simply takes any matching with m edges, ignoring the weights (i.e., matches every job somewhere) in this graph. Prove that there exists a matching of size m.

Hint: You may assume without proof that for bipartite graphs if there is a fractional matching of size k then there exists an integral matching of size $\geq k$.

(c) Prove that the rounding algorithm in (b) satisfies that the total processing time on each machine is at most $2\text{OPT}_{\text{guess}}$.

Hint: For each machine i, first prove that the total processing time of jobs matched to its copies $i_2, \ldots, i_{w_i}$ is at most $\text{OPT}_{\text{guess}}$. Now, since the job matched to copy i_1 has processing time at most $\text{OPT}_{\text{guess}}$, deduce that the total processing time matched to all the w_i copies of i is at most $2\text{OPT}_{\text{guess}}$.

§2 (Online Algorithms)

(a) (Two-Stage Matching: Hardness) Suppose there is a bipartite graph $(A \cup B, E)$ whose edge set E is *unknown* and is partitioned into $E^{(1)} \cup E^{(2)}$ by an adversary. In Stage 1, the algorithm is revealed $(A \cup B, E^{(1)})$ and it has to immediately pick a subset $M^{(1)} \subseteq E^{(1)}$ such that $M^{(1)}$ is a matching. (Note that all edges $E^{(1)}$ are simultaneously revealed and not one-by-one as in Online Matching from class.) All unpicked edges $E^{(1)} \setminus M^{(1)}$ now disappear. In Stage 2, the algorithm is revealed the remaining edges $E^{(2)}$ and it has to select a subset of edges $M^{(2)} \subseteq$

$E^{(2)}$ such that $M^{(1)} \cup M^{(2)}$ is a matching. The goal of the algorithm is to maximize its *competitive ratio*: the minimum ratio over all $E^{(1)} \cup E^{(2)}$ of the expected total matching $\mathbb{E}[|M^{(1)} \cup M^{(2)}|]$ and the max offline matching in $(A \cup B, E)$, where the expectation is over any internal randomness of the algorithm. (Note that the Stage 2 decision for the algorithm is simple as it can just compute the max-matching using $E^{(2)}$ on the vertices that were unmatched after Stage 1.)

Prove that no 2-stage online algorithm can obtain $> 2/3$ competitive ratio.

Hint: In Yao's lemma, consider a distribution over two bipartite graphs where first stage $E^{(1)}$ consists only of a single edge (a, b).

(b) (Online Set Cover) Consider the Online Fractional Set Cover problem studied in class. Let $f = \max_{e \in U} |S \in \mathcal{S} \ : \ e \in S|$ denote the maximum number of sets in which an element could appear. Improve the analysis from class to obtain an $O(\log f)$ competitive algorithm. (This is much better than $O(\log m)$ for $f \ll m$.)

§3 (Online Matching for Random Arrival) Consider a bipartite graph on n agents and m items. Each agent i has a value $v_{ij} \in \mathbb{R}_{\geq 0}$ for item j. The maximum-weight matching problem is to find an assignment $M : [n] \to [m]$ to maximize $\sum_{i \in [n]} v_{iM(i)}$ such that no item j is assigned to more than one agent, i.e., $|M^{-1}(j)| \leq 1$ for all $j \in [m]$. In the online setting, the m items are given up-front and the n agents arrive one by one in a uniformly-random order. Upon arriving, agent i reveals their valuations v_{ij} for $j \in [m]$, whereupon we may irrevocably allocate/assign one of the remaining/unmatched items to i. Let $V^\star$ denote the value of the optimal matching. (The case of $m = 1$ with a single item is exactly the single-item secretary problem.) Consider the following online matching algorithm:

Ignore the first n/e agents. When agent $i \in (n/e, n]$ arrives:

(i) Compute a max-value matching $M^{(i)}$ for the first i arrivals (ignoring past decisions).

(ii) If $M^{(i)}$ matches the current agent i to item j, and if j is still available/unmatched, then allocate j to agent i; else, give nothing to agent i.

(We assume that the matching $M^{(i)}$ is unique, depends only on the identities of the first i requests, and is independent of their arrival order. This can be ensured by randomly perturbing all valuations v_{ij} slightly.)

(a) Prove that in $M^{(i)}$ the i^{th} agent gets expected value at least $V^\star/n$.
Hint: First prove that $M^{(i)}$ has an expected value at least $(i/n)V^\star$.

(b) Similar to the single-item secretary analysis in the class, prove that if agent i is matched to item j in $M^{(i)}$, then j is free with probability $\approx \frac{n}{ei}$. Deduce that the algorithm gives an e-approximation.

Hint: Start by conditioning on the random set of first i agents (not their relative order) and the identity of the i-th agent. Now prove that the probability that j is available for agent i is at least $\prod_{n/e<k<i} \left(1 - \frac{1}{k}\right) \approx \frac{n}{ei}$.

§4 (Stochastic Bandits) Consider a problem where you are given n labeled coins with some *unknown* biases $p_1, \ldots, p_n \in [0,1]$ (i.e., the i-th coin shows Heads independently with some unknown but fixed probability p_i). Consider a T step game where in each step $t \leq T$ you may choose any one of the n coins to toss. (Think of $T \gg n$.) Your goal is to design a tossing strategy to maximize the total number of seen heads. In particular, if the algorithm tosses coin i_t the t-th step and the most biased coin has probability p_{i^*}, then we want to minimize the *regret*:

$$T \cdot p_{i^*} - \textstyle\sum_{t=1}^{T} p_{i_t}.$$

We could run the adversarial bandits algorithm (Exp3) from class to get $O(\sqrt{nT \ln T})$ regret (with $1 - 1/\text{poly}(T)$ probability), but in this exercise we will obtain an exponentially better dependence on T using reward stochasticity (i.e., fixed coin biases).

Let $n_{i,t}$ denote the number of times coin i has been tossed before step t by our algorithm. Define $\hat{p}_{i,t} := \frac{\#\text{ times } i \text{ showed heads before } t}{n_{i,t}}$ to be the empirical bias of coin i before step t.

(a) A natural "greedy" algorithm is to toss each of the n coins once in the first n steps. Then in each subsequent step t, the algorithm plays the coin with the highest empirical bias, i.e., $\arg\max_t \hat{p}_{i,t}$. Prove that for $n = 2$, there exists unknown biases p_1, p_2 such that this algorithm gives $\Omega(T)$ regret with $\Omega(1)$ probability.

Given that the above greedy algorithm could give a large regret, we will now construct an "Optimistic Greedy" algorithm and show that its regret is $O\Big(\sum_{i \neq i^*} \frac{\ln T}{\Delta_i} + \Delta_i\Big)$ with high probability, where $\Delta_i := p_{i^*} - p_i$.

Optimistic Greedy Algorithm:

(i) In the first n steps/rounds, toss each of the n coins once in an arbitrary order.

(ii) In each subsequent step $t \in \{n+1, \ldots, T\}$, toss the coin with the highest "optimistic empirical mean" $\arg\max_i (p_{i,t}^{optim})$ where $p_{i,t}^{optim} := \hat{p}_{i,t} + 10\sqrt{\frac{\ln T}{n_{i,t}}}$.

(b) Consider any fixed coin i. Prove that for this coin the probability that $p_{i,t}^{optim} < p_i$ at any time t during the executing of the algorithm is extremely small, like $< 1/T^5$. Deduce by union bound that we have with $1 - 1/T^4$ probability that for every coin $p_{i,t}^{optim} \geq p_i$ for all t.

(c) Consider any fixed coin $i \neq i^*$. Prove that you will toss this coin at most $O(\frac{\ln T}{\Delta_i^2})$ times, after which you will have $p_{i,t}^{optim} < p_{i^*} \leq p_{i^*,t}^{optim}$ with at least $1 - 1/T^3$ probability (i.e., you would have found that i is a suboptimal coin).

(d) Since each time we toss a suboptimal coin $i \neq i^*$ we incur a regret of Δ_i, deduce from the above parts that the total regret of our algorithm is $O\Big(\sum_{i \neq i^*} \frac{\ln T}{\Delta_i} + \Delta_i\Big)$ with $1 - 1/\text{poly}(T)$ probability.

Remark: The above regret bound $O\big(\sum_{i\neq i^*} \frac{\ln T}{\Delta_i} + \Delta_i\big)$ might appear very weak when even one of the Δ_i is small. It's possible to modify the above analysis to get regret bound $O\Big(\min\big\{\sqrt{nT\ln T}\ ,\ \sum_{i\neq i^*} \frac{\ln T}{\Delta_i} + \Delta_i\big\}\Big)$.

§5 (Approximate LP Solving via Multiplicative Weights) This exercise develops an algorithm to approximately solve Linear Programs.

Consider the problem of finding if a system of linear inequalities as below admits a solution - i.e., whether the system is feasible. This is an example of a feasibility linear program and while it appears restrictive, one can use it solve arbitrary linear programs to obtain approximate solutions. **For all subparts, you may assume that $|a_{ij}| \leq 1$ and $|b_i| \leq 1$ for all i, j.**

$$\begin{aligned} a_1^\top x &\geq b_1 \\ a_2^\top x &\geq b_2 \\ &\vdots \\ a_m^\top x &\geq b_m \\ x_i &\geq 0 \ \forall\ i \in [n] \\ \sum_{i=1}^n x_i &= 1. \end{aligned} \tag{1}$$

(a) Design a simple algorithm to solve the following linear program, which has only two non-trivial constraints. Below, the weights $w_1, w_2, \ldots, w_m$ are fixed (along with the vectors $a_j^\top$ and numbers b_j), and $x_1, \ldots, x_n$ are the variables.

$$\begin{aligned} \max \sum_{j=1}^m w_j(a_j^\top x - b_j) & \\ x_i \geq 0\ \forall i \in [n] & \\ \sum_{i=1}^n x_i = 1. & \end{aligned} \tag{2}$$

(b) Prove that if there exist non-negative weights $w_1, w_2, \ldots, w_m$ such that the value of the program above is negative, then the system (1) is infeasible.

(c) The above setting of finding weights that certify infeasibility of (1) might remind you of the setting of weighting the experts via multiplicative weights update rule discussed in the class. Use these ideas to obtain an algorithm that a) either finds a set of non-negative weights certifying infeasibility of LP in (1) or b) finds a solution x that approximately satisfies all the constraints in (1), i.e., for each $1 \leq j \leq m$, $a_j^\top x - b_j \geq -\epsilon$, and for each $1 \leq i \leq n$, $x_i \geq 0$, and $\sum_{i=1}^n x_i = 1$. Prove that your algorithm terminates after solving $O(\ln(m)/\varepsilon^2)$ LPs of form (2) (you do not need to analyze the remaining runtime).

(Hint: Identify m "experts" - one for each inequality constraint in (1) and maintain a weighting of experts (starting with the uniform weighting of all 1s, say)

for times $t = 0, 1, \ldots,$ - these are your progressively improving guesses for the weights. Solve (2) using the weights at time t. If the value of (2) is negative, you are done, otherwise think of the "cost" of the j^{th} expert as $a_j^\top x^{(t)} - b_j$ where $x^{(t)}$ is the solution to the LP (2) at time t and update the weights.)

Extra Credit:

§6 (extra credit: Two-Stage Matching: Algorithm) Consider the the two-stage matching problem described in §2a. Now we will design a 2/3 competitive algorithm for this problem in the special case where $E^{(1)}$ consists of a perfect matching on the induced vertices.

Let $A^{(1)} \subseteq E^{(1)}$ denote a perfect matching on the induced vertices (i.e., all vertices that have an edge incident to them in $E^{(1)}$ are matched in $A^{(1)}$ but a vertex with no incident edge in $E^{(1)}$ is unmatched).

(i) With probability 2/3 we choose $M^{(1)} = A^{(1)}$ and otherwise we choose $M^{(1)} = \emptyset$. (That is, w.p. 2/3 we select the perfect matching and w.p. 1/3 we pick no edge in Stage 1.)

(ii) In Stage 2, we optimally add edges $M^{(2)} \subseteq E^{(2)}$ to $M^{(1)}$ s.t. $M^{(1)} \cup M^{(2)}$ is a matching.

Prove that this algorithm is 2/3 competitive when $E^{(1)}$ contains a perfect matching $A^{(1)}$ on the induced vertices.

Hint: Construct a randomized dual solution y_u for $u \in A \cup B$ with the same value as the algorithm (i.e., $\sum_e x_e = \sum_u y_u$) and which 2/3-approximately satisfies all the dual constraints (i.e., $y_u + y_v \geq 2/3$ for every edge $(u, v) \in E$).

Remark: It's possible to design a 2/3-competitive algorithm for this problem even when $E^{(1)}$ does not contain a perfect matching on the induced vertices. However, that result is more complicated and uses ideas such as Edmonds-Gallai decomposition.