PRINCETON UNIV FALL '20 COS 521:ADVANCED ALGORITHMS

Homework 2

Out: *Sep 18* Due: *Oct 2*

Instructions:

- Upload your solutions (to the non-extra-credit) to each problem as a **separate PDF** file (one PDF per problem) to codePost. Please make sure you are uploading the correct PDF! Please anonymize your submission (i.e., do not list your name in the PDF), but if you forget, it's OK.
- If you choose to do extra credit, upload your solution to the extra credits as a single separate PDF file to codePost. Please again anonymize your submission.
- You may collaborate with any classmates, textbooks, the Internet, etc. Please upload a brief "collaboration statement" listing any collaborators as a separate PDF on codePost (if you forget, it's OK). But always **write up your solutions individually**.
- 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.
- Each problem is worth twenty points (even those with multiple subparts), unless explicitly stated otherwise.

Problems:

§1 Suppose that you are allowed only a single pass over a large document of English words. Your goal is to (approximately) calculate the number of words n in this document using very little space. The naive algorithm which maintains a counter of the number of seen words uses $O(\log n)$ space (bits).

  (a) Consider an algorithm that maintains a counter X, initialized to 0, and on seeing the next word increases X by 1 with probability $\frac{1}{2^X}$ (otherwise, does nothing and move to the next word). Show that at the end of the stream, $\mathbb{E}[2^X] = n+1$ and $\mathbb{E}[2^{2X}] = \frac{3}{2}n^2 + \frac{3}{2}n + 1$. (Hint: You may want to use induction.)

  (b) Assuming Part (a), show for $0 < \epsilon < 1/2$ how we can use $O(\frac{1}{\epsilon^2} \cdot \text{loglog}\, n)$ space to output with probability at least 0.9 a number $\hat{n} \in [\frac{n}{1+\epsilon}, n(1+\epsilon)]$.

§2 Recall the max-flow problem from undergraduate algorithms: for a directed graph $G(V,E)$ with non-negative capacities c_e for every $e \in E$ and two special vertices s (source, with no incoming edges) and t (sink, with no outgoing edges), a *flow* in G is an assignment $f : E \to \mathbb{R}_{\geq 0}$ such that $f(e) \leq c_e$ for every edge and for every vertex

$v \in V \setminus \{s,t\}$, the total incoming flow $\sum_{(u,v)\in E} f((u,v))$ equals the total outgoing flow $\sum_{(v,w)\in E} f((v,w))$. The task is to find a maximum flow f i.e., a flow f such that $\sum_{(s,u)\in E} f((s,u))$ is maximized.

(a) Show that the following LP is a valid formulation for computing the value of the maximum flow in G. There is a variable $f((u,v))$ for all $(u,v) \in E$. (Hint: below, the inequality between the flows is not a typo. You should show that it is w.l.o.g. to replace the equality with inequality, as this will make it easier to reason about later parts.)

$$\begin{aligned}
&\max \sum_u f((u,t)) \\
&\quad \forall e = (u,v) \in E,\ f((u,v)) \le c_e \\
&\quad \forall v \notin \{s,t\},\ \sum_{(u,v)\in E} f((u,v)) \ge \sum_{(v,w)\in E} f((v,w)) \\
&\quad \forall e \in E,\ f(e) \ge 0
\end{aligned} \tag{1}$$

(b) Write the dual for the LP (1). Show that this dual LP computes the minimum *fractional* s-t cut in G (a cut that separates s and t in G and minimizes the sum of the capacities c_e of the edges going across it. You will know what a fractional s-t cut is once you take the dual: every node isn't entirely on the s side or the t side, but rather partially on each). Use strong LP duality to conclude the *fractional* max-flow min-cut theorem. That is, if the max-flow is C, there exists a fractional s-t cut of value C, and no fractional s-t cut of value $< C$.

(c) Devise a rounding scheme that takes as input a fractional min-cut of value C and outputs a true (deterministic) min-cut of value C. (Hint: there is a simple rounding scheme that works, but it is not a rounding scheme we have already seen in class.)

§3 (Firehouse location) Suppose we model a city as an m-point finite metric space with $d(x,y)$ denoting the distance between points x, y. These $\binom{m}{2}$ distances (which satisfy triangle inequality) are given as part of the input. The city has n houses located at points $v_1, v_2, \ldots, v_n$ in this metric space. The city wishes to build k firehouses and asks you to help find the best locations $c_1, c_2, \ldots, c_k$ for them, which can be located at any of the m points in the city. The *happiness* of a town resident with the final locations depends upon his distance from the closest firehouse. So you decide to minimize the cost function $\sum_{i=1}^n d(v_i, u_i)$ where $u_i \in \{c_1, c_2, \ldots, c_k\}$ is the firehouse closest to v_i. Describe an LP-rounding-based algorithm that runs in *poly*(m) time and solves this problem approximately. If OPT is the optimum cost of a solution with k firehouses, your solution is allowed to use $O(k \log n)$ firehouses and have cost at most OPT.[1] Specifically, you should design an algorithm which runs in polynomial time, and uses $O(k \log n)$ firehouses in expectation, and also has cost at most OPT in expectation.[2]

[1]The term for an approximation guarantee like this is *resource augmentation* — the solution is as good as the optimum, but it requires additional firehouses.

[2]You may want to briefly think about how to modify your solution to run in expected polynomial time

§4 (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.**

$$\begin{aligned} a_1^\top x &\geq b_1 \\ a_2^\top x &\geq b_2 \\ &\vdots \\ a_m^\top x &\geq b_m \\ x_i &\geq 0 \;\forall\; i \in [n] \\ \sum_{i=1}^n x_i &= 1. \end{aligned} \tag{2}$$

(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.

$$\begin{aligned} \max \sum_{j=1}^m w_j (a_j^\top x - b_j) \\ x_i \geq 0 \;\forall i \in [n] \\ \sum_{i=1}^n x_i = 1. \end{aligned} \tag{3}$$

(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 (2) is infeasible.

(c) The above setting of finding weights that certify infeasibility of (2) 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 (2) or b) finds a solution x that approximately satisfies all the constraints in (2), 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 (3) (you do not need to analyze the remaining runtime).

(Hint: Identify m "experts" - one for each inequality constraint in (2) and maintain a weighting of experts (starting with the uniform weighting of all 1s, say)

and use $O(k \log n)$ firehouses with probability one, or how to run in expected polynomial time and guarantee a solution with cost $(1+\varepsilon)$OPT with probability one (or both). But you do not need to write this for full credit.

for times $t = 0, 1, \ldots,$ - these are your progressively improving guesses for the weights. Solve (3) using the weights at time t. If the value of (3) 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 (3) at time t and update the weights.)

§5 (extra credit) In a *combinatorial auction* there are n bidders and m items. Bidder i has a monotone valuation function $v_i(\cdot)$ where $v_i(S)$ denotes their value for set S of items (and $v_i(S \cup T) \geq v_i(S)$ for all S, T). A *Walrasian Equilibrium* is a price for each item $\vec{p}$ such that:

- Each buyer i selects to purchase a set $B_i \in \arg\max_S \{v_i(S) - \sum_{j \in S} p_j\}$.
- The sets B_i are disjoint, and $\cup_i B_i = [m]$.

Prove that a Walrasian equilibrium exists for $v_1, \ldots, v_n$ if and only if the optimum of the LP relaxation below (called the *configuration LP*) is achieved at an integral point (i.e. where each $x_{i,S} \in \{0, 1\}$). Hint: use strong duality!

$$\begin{aligned} \max \sum_i \sum_S v_i(S) \cdot x_{i,S} \\ \forall i, \ \sum_S x_{i,S} = 1 \\ \forall j, \ \sum_{S \ni j} \sum_i x_{i,S} \leq 1 \end{aligned}$$

Also, come up with an example of two valuation functions v_1, v_2 over two items where a Walrasian equilibrium doesn't exist.