Princeton Univ Fall '23	COS 521: Advanced Algorithms
Homework 2	
Out: *Sep 25*	Due: *Oct 8*

Instructions:

- Upload your non-extra solutions to Gradescope in a single PDF file, and mark your solution to each problem. 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 Gradescope. 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 Gradescope (if you forget, it's OK). But always **write up your solutions individually**.
- For each problem, you should have a solid writeup that clearly states key, concrete lemmas towards your full solution (and then you should prove those lemmas). A reader should be able to read any definitions, plus your lemma statements, and quickly conclude from these that your outline is correct. This is the most important part of your writeup, and the precise statements of your lemmas should tie together in a correct logical chain.
- A reader should also be able to verify the proof of each lemma statement in your outline, although it is OK to skip proofs that are clear without justification (and it is OK to skip tedious calculations). Expect to learn throughout the semester what typically counts as 'clear'.
- You can use the style of Lecture Notes and Staff Solutions as a guide. These tend to break down proofs into roughly the same style of concrete lemmas you are expected to do on homeworks. However, they also tend to prove each lemma in slightly more detail than is necessary on PSets (for example, they give proofs of some small claims/observations that would be OK to state without proof on a PSet).
- Each problem is worth twenty points (even those with multiple subparts), unless explicitly stated otherwise.

Problems:

§1 The Ellipsoid algorithm we saw in the lecture solves convex programs assuming a separation oracle. Here, we want to show the opposite. To be more specific, consider the following two tasks regarding a convex body $\mathcal{K}$:

- $\mathsf{OPTIMIZE}(\mathcal{K})$: given a vector $c \in \mathbb{R}^n$, output $\arg\max_{x \in \mathcal{K}} c^\top x$;
- $\mathsf{SEPARATE}(\mathcal{K})$: given a point $x \in \mathbb{R}^n$, output either $x \in \mathcal{K}$, or a separating hyperplane.

We are going to show that if for a specific convex body $\mathcal{K}$, there is a polynomial time algorithm for $\mathsf{OPTIMIZE}(\mathcal{K})$, then there is a polynomial time algorithm for $\mathsf{SEPARATE}(\mathcal{K})$.

(a) Suppose for a given x, we can solve the following LP with *infinitely many* constraints (finding the optimal w and T), then we solve $\mathsf{SEPARATE}(\mathcal{K})$.

$$\begin{aligned} \text{Variables: } & w \in \mathbb{R}^n, T \in \mathbb{R} \\ \text{Maximize: } & w^\top x - T \\ \text{Subject to: } & \forall y \in \mathcal{K}, w^\top y \le T \\ & -1 \le T \le 1 \end{aligned}$$

(b) Design a polynomial time separation oracle for the above LP using $\mathsf{OPTIMIZE}(\mathcal{K})$, and conclude.

§2 In class, we showed that by doing a binary search on the optimal value of an LP, optimizing the objective value can be reduced to deciding feasibility, which is then solvable by the Ellipsoid algorithm. Show how to modify Ellipsoid so that it solves the optimization problem directly, i.e., this reduction is in fact not necessary.

Hint: Maintain the invariant that the optimal solution is contained in the current ellipsoid.

Note: For this problem, you do *not* need to worry about bounding boxes, bit complexity, precision, etc. For example, it suffices that the algorithm finds some x that is $2^{-\Theta(n)}$-close to the true optimum in distance. Also, you do *not* need to worry about the feasible region having dimension less than n, and you may assume the volume of the feasible region is at least $2^{-\Theta(n)}$. You also don't need to reprove the lemmas used in Ellipsoid.

§3 In class we designed a 3/4-approximation for MAX-2SAT using LP rounding. The MAX-SAT problem is similar except for the fact that the clauses can contain any number of literals. Formally, the input consists of n boolean variables $x_1, x_2, \ldots, x_n$ (each may be either 0 (false) or 1 (true)), m clauses $C_1, C_2, \ldots, C_m$ (each of which consists of disjunction (an "or") of some number variables or their negations) and a non-negative weight w_i for each clause. The objective is to find an assignment of 1 or 0 to x_is that maximize the total weight of satisfied clauses. As we saw in the class, a clause is satisfied if one of its non-negated variable is set to 1, or one of the negated

variable is set to 0. You can assume that no literal is repeated in a clause and at most one of x_i or $\neg x_i$ appears in any clause.

(a) Generalize the LP relaxation for MAX-2SAT seen in the class to obtain a LP relaxation of the MAX-SAT problem.

(b) Use the standard randomized rounding algorithm (the same one we used in class for MAX-2SAT) on the LP-relaxation you designed in part (1) to give a $(1-1/e)$ approximation algorithm for MAX-SAT. Recall that clauses can be of any length. (Hint: there is a clean way to resolve "the math" without excessive calculations).

(c) A naive algorithm for MAX-SAT problem is to set each variable to true with probability 1/2 (without writing any LP). It is easy to see that this *unbiased randomized* algorithm of MAX-SAT achieves 1/2-approximation in expectation. Show the algorithm that returns the best of two solutions given by the randomized rounding of the LP and the simple unbiased randomized algorithm is a 3/4-approximation algorithm of MAX-SAT. (Hint: it may help to realize that in fact *randomly* selecting one of these two algorithms to run also gives a 3/4-approximation in expectation).

(d) Using the previous part (and in particular, the hint) for intuition, design a direct rounding scheme of your LP relaxation to get a 3/4-approximation (that is, design a function $f(\cdot)$ which assigns a literal x_i to be true independently with probability $f(z)$ when the corresponding variable z_i in your LP relaxation is equal to z). (Hint: here, it may get messy to fully resolve the calculations. You will get full credit if you state the correct rounding scheme and clearly state the necessary inequalities for the proof. You should also attempt to show that the inequalities hold for your own benefit, but not for full credit).

§4 (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 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.

§5 (Discrepancy Theory) Suppose you are given a matrix $A \in \{0,1\}^{n \times n}$ such that each column of A has at most s ones. Our goal is to *color* each column with -1 or $+1$ so that when we take a linear combination of the columns (multiplied by their color), we get a vector where each coordinate is $O(s)$. That is, we are hoping to color each column in a way so that all rows have a nearly-equal number of zeroes and ones.

Formally, we seek a vector (coloring) $\vec{\varepsilon} \in \{-1,+1\}^n$ such that the *discrepancy* $\|A\vec{\varepsilon}\|_\infty = O(s)$. Observe that if $\vec{a}_t$ is the t-th column of A, we can equivalently write: $\|A\vec{\varepsilon}\|_\infty := \|\sum_{t=1}^n \vec{a}_t \varepsilon_t\|_\infty := \max_{i=1}^n \{|a_{ti}\varepsilon_t|\} = \max_{i=1}^n \{|A_{it}\varepsilon_t|\}$.

We will analyze the following algorithm to achieve this task. The algorithm will iteratively color more and more columns of A. Initially, all columns are "uncolored" and a column $\vec{a}_t$ gets "colored" when ε_t is defined.

- In any iteration, if the number of remaining uncolored columns is at most $2s$, color them arbitrarily and halt.
- Otherwise, we solve the following feasibility LP.

$$\begin{aligned} \text{Variables: } & x_t \ \forall\ t \in [n]. \\ \text{Constraints: } & x_t \in [-1,1] \ \forall \text{ uncolored } t. \\ & x_t = \varepsilon_t \ \forall \text{ previously colored } t. \\ & \sum_t A_{it} x_t = 0 \ \forall\ i \text{ such that row } i \text{ has } \geq 2s \text{ uncolored ones.} \end{aligned}$$

  That is, we are allowed to fractionally color each column in $[-1,1]$. However, we must respect any previously colored columns. Moreover, if any row has many uncolored ones, we must ensure that the (fractional) coloring results in a 0 for that row.
- Find a *basic feasible solution* x^* of the above LP, and for any uncolored column t that gets $x_t^* \in \{-1,+1\}$ in this basic solution, we permanently set its ε_t to that color. A *basic feasible solution* of an LP is one that is an extreme point of the feasible space. That is, it cannot be written as a convex combination of other feasible points. You may use without proof that any LP with a feasible solution has at least one basic feasible solution. You may also use without proof that any basic feasible solution of the above LP makes exactly n constraints tight (hold with equality).[3]
- Repeat.

(a) Prove that in every iteration, the LP has a feasible solution.

(b) Prove that in every iteration, the number of row constraints in the LP is at most half the number of currently uncolored columns.

Conclude that a basic feasible solution will integrally color at least half of the uncolored columns in each iteration.

[3]Such a solution can be found by picking a uniformly random vector $\vec{c}$, and optimizing $\vec{c} \cdot \vec{x}$ over the feasible region.

(c) Prove that this algorithm obtains a solution with discrepancy $O(s)$.

Remark: There is nothing special about $A \in \{0,1\}^{n \times n}$ vs. $A \in \{0,1\}^{n \times T}$, i.e., having $T \gg n$ columns. We can easily extend the above result to T columns using the idea of finding a basic solution in the beginning with at most n uncolored columns.

Extra Credit:

§1 (extra credit) Consider the following problem: there are $n > k$ independent (but not identically distributed) non-negative random variables $X_1, \ldots, X_n$ drawn according to distributions $D_1, \ldots, D_n$. Initially, you know each D_i but none of the X_is.

Starting from $i = 1$, each X_i is revealed one at a time. Immediately after it is revealed, you must decide whether to "accept i" or "reject i," before seeing the next X_{i+1}. You may accept at most k elements in total (that is, once you've accepted k times, you must reject everything that comes after). Your reward at the end is $\sum_{i|i \text{ was accepted}} X_i$.

(a) For general k, design a policy that guarantees expected reward at least $(1 - O(\sqrt{\frac{\ln(k)}{k}})) \cdot \mathbb{E}_{X_1,\ldots,X_n \leftarrow D_1,\ldots,D_n}[\sum_{j=1}^{k} X_{r(j)}]$, where r is a permutation from $[n]$ to $[n]$ satisfying $X_{r(1)} \geq X_{r(2)} \geq \ldots \geq X_{r(n)}$ (i.e. the policy gets expected reward at least $(1 - O(\sqrt{\frac{\ln(k)}{k}}))$ times the expected sum of top k weights, which is the best you could do even if you knew all the weights up front).[4]

Hint: Try to set up a simple policy that can be analyzed using a Chernoff bound.

(b) Come up with an example showing that it is not possible to improve the above guarantee beyond $(1 - \Omega(1/\sqrt{k}))$ (which is optimal - no need to prove this).

Hint: an example exists whose complete proof should fit in half a page. You may use without proof the fact that if X is the number of coin flips which land heads from k independent fair coin flips, then $\mathbb{E}[|X - k/2|] = \Theta(\sqrt{k})$.

[4] For simplicity of exposition, you may assume that each random variable is continuous. You may want to think about how to adapt your analysis in case the random variables are discrete, but do not need to write this in your proof for full credit.