CS261: Problem Set #4

Due by 11:59 PM on Tuesday, June 2, 2015

Instructions:

(1) Form a group of 1-3 students. You should turn in only one write-up for your entire group.

(2) Turn in your solutions by email to `cs261submissions@gmail.com`. Please type your solutions if possible and feel free to use the LaTeX template provided on the course home page.

(3) Write convincingly but not excessively.

(4) Some of these problems are difficult, so your group may not solve them all to completion. In this case, you can write up what you've got (subject to (3), above): partial proofs, lemmas, high-level ideas, counterexamples, and so on.

(5) Except where otherwise noted, you may refer to your course notes and the specific supplementary readings listed on the course Web page *only*. You can also review any relevant materials from your undergraduate algorithms course.

(6) You can discuss the problems verbally at a high level with other groups. And of course, you are encouraged to contact the course staff (via Piazza or office hours) for additional help.

(7) If you discuss solution approaches with anyone outside of your group, you must list their names on the front page of your write-up.

(8) Refer to the course Web page for the late day policy.

(9) Refer to Piazza for the course regrade policy.

Problem 19

[ESTIMATED DIFFICULTY LEVEL: Somewhat difficult.] This problem gives a different $O(\log n)$-approximation algorithm for the Set Cover problem, using randomized rounding. Recall from Lecture #13 our linear programming relaxation of the problem (with nonnegative set costs):

$$\min \sum_{j=1}^{m} c_j x_j$$

subject to

$$\sum_{j\,:\,i \in S_j} x_j \geq 1 \qquad \text{for all elements } i \in U$$

and

$$x_j \geq 0 \qquad \text{for every sets } S_j.$$

Consider the following algorithm:

1. Solve the linear program to obtain an optimal solution $\mathbf{x}^*$.
2. For $t = 1, \ldots, 2\ln n$:
   (a) Initialize $\mathcal{C}_t = \emptyset$.

(b) Independently for $j = 1, \ldots, m$:

  i. With probability x_j^*, add S_j to $\mathcal{C}_t$.

3. Return $\mathcal{C} = \cup_{t=1}^{2\ln n} \mathcal{C}_t$.

(a) Prove that the expected cost of the output of the algorithm is at most $2 \ln n$ times the cost of an optimal set cover.

[Hint: linearity of expectation.]

(b) Prove that for every element $i \in U$ and every $t = 1, 2, \ldots, 2\ln n$, the probability that i belongs to a set of $\mathcal{C}_t$ is at least $1 - \frac{1}{e}$.

[Hint: Use that $1 - y \le e^{-y}$ for all y.]

(c) Prove that the algorithm outputs a feasible set cover with probability at least $1 - \frac{1}{n}$.

(d) The algorithm above is a "Monte Carlo" randomized algorithm, meaning that it always runs in polynomial time but sometimes fails to produce a feasible solution. Explain how to convert it to a "Las Vegas" randomized algorithm, which is guaranteed to output a feasible set cover with cost $O(\log n)$ times the minimum possible, but which only runs in polynomial time in expectation.

[Hints: first use Markov's inequality (Exercise 33) and the Union Bound to argue that there is a large probability that the algorithm outputs a feasible solution with near-optimal cost. Run this algorithm a polynomial number of times; if it fails to produce a near-optimal feasible solution every single time, then resort to brute-force search.]

Problem 20

[ESTIMATED DIFFICULTY LEVEL: Somewhat difficult.] The goal of this problem is to introduce a generalization of the multiway cut problem (see Problem 17) and use randomized linear programming rounding to give a good approximation for it.

In the *uniform labeling problem*, we are given an undirected graph $G = (V, E)$, costs $c_e \ge 0$ for all edges $e \in E$, and a set L of labels that can be assigned to the vertices of V. There is a non-negative cost $c_v^i \ge 0$ for assigning label $i \in L$ to vertex $v \in V$, and the edge cost c_e is incurred if and only if e's endpoints are given distinct labels. The goal of the problem is to assign each vertex a label so as to minimize the total cost.[1]

(a) Explain why the minimum multiway cut problem is a special case of the uniform labeling problem.

(b) Prove that the following is a linear programming relaxation of the problem:

$$\min \frac{1}{2} \sum_{e \in E} c_e \sum_{i \in L} z_e^i + \sum_{v \in V} \sum_{i \in L} c_v^i x_v^i$$

subject to

$$\sum_{i \in L} x_v^i = 1 \qquad \text{for all } v \in V$$

$$z_e^i \ge x_u^i - x_v^i \qquad \text{for all } e = (u,v) \in E \text{ and } i \in L$$

$$z_e^i \ge x_v^i - x_u^i \qquad \text{for all } e = (u,v) \in E \text{ and } i \in L$$

$$z_e^i \ge 0 \qquad \text{for all } e \in E \text{ and } i \in L$$

$$x_v^i \ge 0 \qquad \text{for all } v \in V \text{ and } i \in L.$$

Specifically, prove that for every feasible solution to the uniform labeling problem, there is a corresponding 0-1 feasible solution to this linear program that has the same objective function value.

[1]The motivation for the problem comes from image segmentation, generalizing the foreground-background segmentation problem discussed in Lecture #4.

(c) Consider now the following algorithm. First, the algorithm solves the linear programming relaxation above. The algorithm then proceeds in phases. In each phase, it picks a label $i \in L$ uniformly at random, and independently a number $\alpha \in [0,1]$ uniformly at random. For each vertex $v \in V$ that has not yet been assigned a label, if $\alpha \le x_v^i$, then we assign v the label i (otherwise it remains unassigned).

To begin the analysis of this randomized rounding algorithm, consider the start of a phase and suppose that the vertex $v \in V$ has not yet been assigned a label. Prove that (i) the probability that v is assigned the label i in the current phase is exactly $x_v^i/|L|$; and (ii) the probability that it is assigned some label in the current phase is exactly $1/|L|$.

(d) Prove that the algorithm assigns the label $i \in L$ to the vertex $v \in V$ with probability exactly x_v^i.

(e) We say that an edge e is *separated by a phase* if both endpoints were not assigned prior to the phase, and exactly one of the endpoints is assigned a label in this phase. Prove that, conditioned on neither endpoint being assigned yet, the probability that an edge e is separated by a given phase is at most $\frac{1}{|L|}\sum_{i\in L} z_e^i$.

(f) Prove that, for every edge e, the probability that the algorithm assigns different labels to e's endpoints is at most $\sum_{i\in L} z_e^i$.

[Hint: it might help to identify a sufficient condition for an edge $e = (u,v)$ to *not* be separated, and to relate the probability of this to the quantity $\sum_{i\in L} \min\{x_u^i, x_v^i\}$.]

(g) Prove that the expected cost of the solution returned by the algorithm is at most twice the cost of an optimal solution.

Problem 21

[ESTIMATED DIFFICULTY LEVEL: Not too bad.] The goal of this problem is to give some more examples of NP-hard problems for which a trivial randomized algorithm already achieves a constant-factor approximation, with the analysis being a simple application of the linearity of expectation.

(a) See Problem 23(a) for the definition of the Max k-Cut problem. Give a trivial randomized algorithm that returns a solution with expected objective function value at least $\frac{k-1}{k}$ times that of an optimal solution.

(b) Consider the maximum-weight independent set problem, where the input is an undirected graph $G = (V,E)$ with nonnegative vertex weights w_v for all $v \in V$, and the goal is to compute an independent set (a subset $S \subseteq V$ with at most one endpoint of each edge) of maximum-possible total weight. Consider the algorithm that (i) orders the vertices $V = v_1, v_2, \dots, v_n$ uniformly at random; and (ii) does a single pass through the vertices in this order, adding a vertex v_i to the independent-set-so-far S if and only if no neighbors of v_i have already been added to S.

Prove that the expected total weight of the independent set computed by this algorithm is at least a $1/(\Delta+1)$ fraction of the maximum possible, where Δ denotes the maximum degree of the graph G.

[Hint: for a given vertex v, identify a sufficient condition on the vertex ordering that guarantees the inclusion of the vertex v into S.]

(c) One of the (many) hard problems that arises in genome mapping can be formulated in the following abstract way. We are given a set of n *markers* $\{\mu_1, \dots, \mu_n\}$ — these are positions on a chromosome that we are trying to map — and our goal is to output a linear ordering of these markers. The output should be as consistent as possible with a set of k "betweenness constraints," each specified by an ordered triple (μ_i, μ_j, μ_k), requiring that μ_j lie between μ_i and μ_k in the total ordering that we produce. (μ_i can precede μ_j and μ_k follow μ_j, or vice versa; both of these are allowed.)

Give a simple randomized algorithm such that the expected number of constraints satisfied by its ordering is at least $\frac{1}{3}$ times the maximum possible.

Problem 22

[ESTIMATED DIFFICULTY LEVEL: A bit difficult.] This problem considers a natural clustering problem, where it's relatively easy to obtain a good approximation algorithm and a matching hardness of approximation bound.

The input to the *metric k-center* problem is the same as that in the metric TSP problem — a complete undirected graph $G = (V, E)$ where each edge e has a nonnegative cost c_e, and the edge costs satisfy the triangle inequality ($c_{uv} + c_{vw} \geq c_{uw}$ for all $u, v, w \in V$). Also given is a parameter k. Feasible solutions correspond to choices of k *centers*, meaning subsets $S \subseteq V$ of size k. The objective function is to minimize the furthest distance from a point to its nearest center:

$$\min_{S \subseteq V \,:\, |S|=k} \max_{v \in V} \min_{s \in S} c_{sv}. \tag{1}$$

We'll also refer to the well-known NP-complete *Dominating Set* problem, where given an undirected graph G and a parameter k, the goal is to decide whether or not G has a dominating set of size at most k.[2]

(a) (No need to hand in.) Let OPT denote the optimal objective function value (1). Observe that OPT equals the cost c_e of some edge, which immediately narrows down its possible values to a set of $\binom{n}{2}$ different possibilities (where $n = |V|$).

(b) Given an instance G to the metric k-center problem, let G_D denote the graph with vertices V and with an edge (u, v) if and only if the edge cost c_{uv} in G is at most $2D$. Prove that if we can efficiently compute a dominating set of size at most k in G_D, then we can efficiently compute a solution to the k-center instance that has objective function value at most $2D$.

(c) Prove that the following greedy algorithm computes a dominating set in G_{OPT} with size at most k:

- $S = \emptyset$
- While S is not a dominating set in G_{OPT}:
  - Let v be a vertex that is not in S and has no neighbor in S — there must be one, by the definition of a dominating set — and add v so S.

[Hint: the optimal k-center solution partitions the vertex set V into k "clusters," where the ith group consists of those vertices for which the ith center is the closest center. Argue that the algorithm above never picks two different vertices from the same cluster.]

(d) Put (a)–(c) together to obtain a 2-approximation algorithm for the metric k-center problem. (The running time of your algorithm should be polynomial in both n and k.)

(e) Using a reduction from the Dominating Set problem, prove that for every $\epsilon > 0$, there is no $(2 - \epsilon)$-approximation algorithm for the metric k-center problem, unless $P = NP$.

[Hint: look to our reduction to TSP (Lecture #12) for inspiration.]

Problem 23

[ESTIMATED DIFFICULTY LEVEL: Somewhat difficult.] This problem explores *local search* as a technique for designing good approximation algorithms.

(a) In the *Max k-Cut* problem, the input is an undirected graph $G = (V, E)$ and a nonnegative weight w_e per edge, and the goal is to partition V into at most k sets such that the sum of the weights of the cut edges — edges with endpoints in different sets of the partition — is as large as possible. The obvious local search algorithm for the problem is:

1. Initialize $(S_1, \ldots, S_k)$ to an arbitrary partition of V.

[2] A *dominating set* is a subset $S \subseteq V$ of vertices such that every vertex $v \in V$ either belongs to S or has a neighbor in S.

2. While there exists an improving move:

   [An *improving move* is a vertex $v \in S_i$ and a set S_j such that moving v from S_i to S_j strictly increases the objective function.]

   (a) Choose an arbitrary improving move and execute it — move the vertex v from S_i to S_j.

Since each iteration increases the objective function value, this algorithm cannot cycle and eventually terminates, at a "local maximum."

Prove that this local search algorithm is guaranteed to terminate at a solution with objective function value at least $\frac{k-1}{k}$ times the maximum possible.

[Hint: prove the statement first for $k = 2$; your argument should generalize easily. Also, you might find it easier to prove the stronger statement that the algorithm's final partition has objective function value at least $\frac{k-1}{k}$ times the sum of all the edge weights.]

(b) It's true (but not easy to prove) that the algorithm in (a) does not always run in polynomial time. Here's a fix: letting $W = \sum_{e \in E} w_e$ denote the sum of the weights of all of the edges and $\epsilon > 0$ a constant, suppose that we only bother to make a local move when the objective function increases by at least $4\epsilon W/n$. Prove that this modified local search algorithm runs in polynomial time and returns a cut with value at least $(\frac{1}{2} - \epsilon)$ times the maximum possible. (Your running time can depend polynomially on $\frac{1}{\epsilon}$.)

(c) Recall the uniform metric labeling problem from Problem 20. We now give an equally good approximation algorithm based on local search.

Our local search algorithm uses the following local move. Given a current assignment of labels to vertices in V, it picks some label $i \in L$ and considers the minimum-cost *i-expansion* of the label i; that is, it considers the minimum-cost assignment of labels to vertices in V in which each vertex either keeps its current label or is relabeled with label i (note that all vertices currently with label i do not change their label). If the cost of the labeling from the i-expansion is cheaper than the current labeling, then we switch to the labeling from the i-expansion. We continue until we find a locally optimal solution; that is, an assignment of labels to vertices such that every i-expansion can only increase the cost of the current assignment.

Give a polynomial-time algorithm that computes an improving i-expansion, or correctly decides that no such improving move exists.

[Hint: recall Lecture #4.]

(d) Prove that the local search algorithm in (c) is guaranteed to terminate at an assignment with cost at most twice the minimum possible.

[Hint: the optimal solution suggests some local moves. By assumption, these are not improving. What do these inequalities imply about the overall cost of the local minimum?]

Problem 24

[ESTIMATED DIFFICULTY LEVEL: A bit difficult.] In this problem we'll show that there is no online algorithm for the online bipartite matching problem with competitive ratio better than $1 - \frac{1}{e} \approx 63.2\%$.

Consider the following probability distribution over online bipartite matching instances. There are n left-hand side vertices L, which are known up front. Let π be an ordering of L, chosen uniformly at random. The n vertices of the right-hand side R arrive one by one, with the ith vertex of R connected to the last $n - i + 1$ vertices of L (according to the random ordering π).

(a) Explain why $OPT = n$ for every such instance.

(b) Consider an arbitrary deterministic online algorithm $\mathcal{A}$. Prove that for every $i \in \{1, 2, \ldots, n\}$, the probability (over the choice of π) that $\mathcal{A}$ matches the ith vertex of L (according to π) is at most

$$\min \left\{ \sum_{j=1}^{i} \frac{1}{n-j+1}, 1 \right\}.$$

[Hint: for example, in the first iteration, assume that $\mathcal{A}$ matches the first vertex of R to the vertex $v \in L$. Note that $\mathcal{A}$ must make this decision without knowing π. What can you say if v does not happen to be the first vertex of π?]

(c) Prove that for every deterministic online algorithm $\mathcal{A}$, the expected (over π) size of the matching produced by $\mathcal{A}$ is at most

$$\sum_{i=1}^{n} \min \left\{ \sum_{j=1}^{i} \frac{1}{n-j+1}, 1 \right\}, \tag{2}$$

and prove that (2) approaches $n(1 - \frac{1}{e})$ as $n \to \infty$.

[Hint: for the second part, recall that $\sum_{j=1}^{d} \frac{1}{j} \approx \ln d$ (up to an additive constant less than 1). For what value of i is the inner sum roughly equal to 1?]

(d) Extend (c) to randomized online algorithms $\mathcal{A}$, where the expectation is now over both π and the internal coin flips of $\mathcal{A}$.

[Hint: use the fact that a randomized online algorithm is a probability distribution over deterministic online algorithms (as flipping all of $\mathcal{A}$'s coins in advance yields a deterministic algorithm).]

(e) Prove that for every $\epsilon > 0$ and (possibly randomized) online bipartite matching algorithm $\mathcal{A}$, there exists an input such that the expected (over $\mathcal{A}$'s coin flips) size of $\mathcal{A}$'s output is no more than $1 - \frac{1}{e} + \epsilon$ times that of an optimal solution.

Problem 25 (Optional — Extra Credit)

[ESTIMATED DIFFICULTY LEVEL: Difficult.] Recall the low-congestion routing problem studied in Lecture #14. The input is a directed graph $G = (V, E)$ with k source-sink pairs $(s_1, t_1), \ldots, (s_k, t_k)$ and a positive integral capacity u_e for each edge. The goal is to pick a single s_i-t_i path P_i for each $i = 1, 2, \ldots, k$ such that the maximum overflow ratio

$$\max_{e \in E} \frac{\#\text{ of } P_i\text{'s that include } e}{u_e} \tag{3}$$

is as small as possible. Recall that the LP relaxation computes a (fractional) s_i-t_i flow f^i for each $i = 1, 2, \ldots, k$, such that each f^i has value 1 and the ratio

$$\max_{e \in E} \frac{\text{total flow on } e}{u_e} \tag{4}$$

is as small as possible.

We showed in Lecture #14 that randomized rounding yields, with high probability, a solution such that the overflow ratio (3) is at most $\frac{3 \ln n}{\ln \ln n}$ times that of the optimal LP objective function value (4) (where $n = |V|$). This implies that the worst-case integrality gap of this LP relaxation is at most $\frac{3 \ln n}{\ln \ln n}$ (why?). Prove a matching lower bound: for arbitrarily large values of n, there are graphs G with n vertices such that the minimum-possible value of (3) for an integral solution is $\Omega(\ln n / \ln \ln n)$ times the optimal objective function value (4) of the LP relaxation.

[Hint: use a recursive construction. If your base case is a network with some number x of vertices, after k iterations of the construction you should have a network with x^k nodes and integrality gap k.]