CS261: Problem Set #4

Due by 11:59 PM on Tuesday, March 8, 2016

Instructions:

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

(2) Submission instructions: We are using Gradescope for the homework submissions. Go to www.gradescope.com to either login or create a new account. Use the course code 9B3BEM to register for CS261. Only one group member needs to submit the assignment. When submitting, please remember to add all group member names in Gradescope.

(3) Please type your solutions if possible and we encourage you to use the LaTeX template provided on the course home page.

(4) Write convincingly but not excessively.

(5) 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.

(6) Except where otherwise noted, you may refer to the course lecture notes *only*. You can also review any relevant materials from your undergraduate algorithms course.

(7) 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.

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

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

Problem 19

This problem considers randomized algorithms for the online (integral) bipartite matching problem (as in Lecture #14).

(a) Consider the following algorithm: when a new vertex $w \in R$ arrives, among the unmatched neighbors of w (if any), choose one uniformly at random to match to w.

Prove that the competitive ratio of this algorithm is strictly smaller than $1 - \frac{1}{e}$.

(b) The remaining parts consider the following algorithm: before any vertices of R arrive, independently pick a number y_v uniformly at random from $[0, 1]$ for each vertex $v \in L$. Then, when a new vertex $w \in R$ arrives, match w to its unmatched neighbor with the smallest y-value (or to no one if all its neighbors are already matched).

For the analysis, when v and w are matched, define $q_v = g(y_v)$ and $q_w = 1 - g(y_v)$, where $g(y) = e^{y-1}$ is the same function used in Lecture #14.

Prove that with probability 1, at the end of the algorithm, $\sum_{v \in L \cup R} q_v$ equals the size of the computed matching.

(c) Fix an edge (v, w) in the final graph. Condition on the choice of y_x for every vertex $x \in L \cup R \setminus \{v\}$ other than v; q_v remains random. As a thought experiment, suppose we re-run the online algorithm from scratch with v deleted (the rest of the input and the y-values stay the same), and let $t \in L$ denote the vertex to which w is matched (if any).

Prove that the conditional expectation of q_v (given q_x for all $x \in L \cup R \setminus \{v\}$) is at least $\int_0^{y_t} g(z)dz$. (If t does not exist, interpret y_t as 1.)

[Hint: prove that v is matched (in the online algorithm with the original input, not in the thought experiment) whenever $y_v < y_t$. Conditioned on this event, what is the distribution of y_v?]

(d) Prove that, conditioned on q_x for all $x \in L \cup R \setminus \{v\}$, $q_w \geq 1 - g(y_t)$.

[Hint: prove that w is always matched (in the online algorithm with the original input) to a vertex with y-value at most y_t.]

(e) Prove that the randomized algorithm in (b) is $(1 - \frac{1}{e})$-competitive, meaning that for every input, the expected value of the computed matching (over the algorithm's coin flips) is at least $1 - \frac{1}{e}$ times the size of a maximum matching.

[Hint: use the expectation of the q-values to define a feasible dual solution.]

Problem 20

A set function $f : 2^U \to \mathbb{R}_+$ is *monotone* if $f(S) \leq f(T)$ whenever $S \subseteq T \subseteq U$. Such a function is *submodular* if it has diminishing returns: whenever $S \subseteq T \subseteq U$ and $i \notin T$, then

$$f(T \cup \{i\}) - f(T) \leq f(S \cup \{i\}) - f(S). \tag{1}$$

We consider the problem of, given a function f and a budget k, computing[1]

$$\max_{S \subseteq U : |S| = k} f(S). \tag{2}$$

(a) Prove that set coverage problem (Lecture #15) is a special case of this problem.

(b) Let $G = (V, E)$ be a directed graph and $p \in [0, 1]$ a parameter. Recall the cascade model from Lecture #15:

- Initially the vertices in some set S are "active," all other vertices are "inactive." Every edge is initially "undetermined."
- While there is an active vertex v and an undetermined edge (v, w):
  - with probability p, edge (v, w) is marked "active," otherwise it is marked "inactive;"
  - if (v, w) is active and w is inactive, then mark w as active.

Let $f(S)$ denote the expected number of active vertices at the conclusion of the cascade, given that the vertices of S are active at the beginning. (The expectation is over the coin flips made for the edges.) Prove that f is monotone and submodular.

[Hint: prove that the condition (1) is preserved under convex combinations.]

(c) Let f be a monotone submodular function. Define the greedy algorithm in the obvious way — at each of k iterations, add to S the element that increases f the most. Suppose at some iteration the current greedy solution is S and it decides to add i to S. Prove that

$$f(S \cup \{i\}) - f(S) \geq \frac{1}{k} \left(OPT - f(S) \right),$$

where OPT is the optimal value in (2).

[Hint: If you added every element in the optimal solution to S, where would you end up? Then use submodularity.]

[1] Don't worry about how f is represented in the input. We assume that it is possible to compute $f(S)$ from S in a reasonable amount of time.

(d) Prove that for every monotone submodular function f, the greedy algorithm is a $(1-\frac{1}{e})$-approximation algorithm.

Problem 21

This problem considers the "$\{1,2\}$" special case of the asymmetric traveling salesman problem (ATSP). The input is a complete directed graph $G=(V,E)$, with all $n(n-1)$ directed edges present, where each edge e has a cost c_e that is either 1 or 2. Note that the triangle inequality holds in every such graph.

(a) Explain why the $\{1,2\}$ special case of ATSP is NP-hard.

(b) Explain why it's trivial to obtain a polynomial-time 2-approximation algorithm for the $\{1,2\}$ special case of ATSP.

(c) This part considers a useful relaxation of the ATSP problem. A *cycle cover* of a directed graph $G=(V,E)$ is a collection $C_1,\ldots,C_k$ of simple directed cycles, each with at least two edges, such that every vertex of G belongs to exactly one of the cycles. (A traveling salesman tour is the special case where $k=1$.) Prove that given a directed graph with edge costs, a cycle cover with minimum total cost can be computed in polynomial time.

[Hint: bipartite matching.]

(d) Using (c) as a subroutine, give a $\frac{3}{2}$-approximation algorithm for the $\{1,2\}$ special case of the ATSP problem.

Problem 22

This problem gives an application of randomized linear programming rounding in approximation algorithms. In the *uniform labeling problem*, we are given an undirected graph $G=(V,E)$, costs $c_e \geq 0$ 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 $c_v^i \geq 0$ 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.[2]

(a) 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

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

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

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

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

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

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.

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

(b) 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 $\alpha \leq x_v^i$, 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 $1/|L|$.

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

(d) 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 $\frac{1}{|L|} \sum_{i \in L} z_e^i$.

(e) 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 $\sum_{i \in L} \min\{x_u^i, x_v^i\}$.]

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

Problem 23

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 for each 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. 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) Recall the uniform metric labeling problem from Problem 22. 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.]

(c) Prove that the local search algorithm in (b) 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

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:

$$\min_{S \subseteq V\,:\,|S|=k} \max_{v \in V} \min_{s \in S} c_{sv}. \tag{3}$$

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.[3]

(a) (No need to hand in.) Let OPT denote the optimal objective function value (3). 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 #16) for inspiration.]

[3] 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.