Homework 6 for CMSC 858E

Due 12/05/2018

Note: problem and page number are for the PDF version provided on course website. In case the problem/section/page numbers are not the same, the corresponding pages are also included in this pdf.

Problem 1

Solve problem 5.1, on page 135 in *The Design of Approximation Algorithms*.

Problem 2

Solve problem 5.3 on page 136 in *The Design of Approximation Algorithms*.

Problem 3

Solve problem 7.5 on page 192 in *The Design of Approximation Algorithms*. You can work in groups for this problem, you are welcomed to solve it by yourself.

Problem 4

Solve problem 7.8 on page 193 in *The Design of Approximation Algorithms*.

Problem 5

Solve problem 8.2 on page 225 in *The Design of Approximation Algorithms*.

Proof. This follows from Lemmas 5.33 and 5.34. The probability that both statements of the lemma are true is at least one minus the sum of the probabilities that either statement is false. The probability that every $v \notin S$ has no neighbor in S is at worst n times the probability that a given vertex $v \notin S$ has no neighbor in S. Since $\delta \leq 1$, the overall probability that both statements are true is at least

$$1 - n^{-c/\delta} - n \cdot n^{-3c} \geq 1 - 2n^{-(c-1)}.$$

□

Now we assume we have some coloring of the vertices in S, not necessarily one that is consistent with the correct coloring of the graph. We also assume that every vertex not in S has at least one neighbor in S. We further assume that the coloring of S is such that every edge with both endpoints in S has differently colored endpoints, since otherwise this is clearly not a correct coloring of the graph. Assume we color the graph with colors $\{0, 1, 2\}$. Given a vertex $v \notin S$, because it has some neighbor in S colored with some color $n(v) \in \{0, 1, 2\}$, we know that v cannot be colored with color $n(v)$. Possibly v has other neighbors in S with colors other than $n(v)$. Either this forces the color of v or there is no way we can successfully color v; in the latter case our current coloring of S must not have been correct, and we terminate. If the color of v is not determined, then we create a binary variable $x(v)$, which if true indicates that we color v with color $n(v) + 1 \pmod 3$, and if false indicates that we color v with color $n(v) - 1 \pmod 3$. Now every edge $(u, v) \in E$ for $u, v \notin S$ imposes the constraint $n(u) \neq n(v)$. To capture this, we create an instance of the maximum satisfiability problem such that all clauses are satisfiable if and only if the vertices not in S can be correctly colored. For each possible setting of the Boolean variables $x(u)$ and $x(v)$ that would cause $n(u) = n(v)$, we create a disjunction of $x(u)$ and $x(v)$ that is false if it implies $n(u) = n(v)$; for example, if $x(u) = \mathsf{true}$ and $x(v) = \mathsf{false}$ implies that $n(u) = n(v)$, then we create a clause $(\overline{x(u)} \vee x(v))$. Since G is 3-colorable, given a correct coloring of S, there exists a setting of the variables $x(v)$ which satisfies all the clauses. Since each clause has two variables, it is possible to determine in polynomial time whether the instance is satisfiable or not; we leave it as an exercise to the reader to prove this (Exercise 6.3). Obviously if we find a setting of the variables that satisfies all constraints, this implies a correct coloring of the entire graph, whereas if the constraints are not satisfiable, our current coloring of S must not have been correct.

In Section 12.4, we'll revisit the idea from this section of drawing a small random sample of a graph and using it to determine the overall solution for the maximum cut problem in dense graphs.

Exercises

5.1 In the *maximum k-cut problem*, we are given an undirected graph $G = (V, E)$, and non-negative weights $w_{ij} \geq 0$ for all $(i, j) \in E$. The goal is to partition the vertex set V into k parts $V_1, \ldots, V_k$ so as to maximize the weight of all edges whose endpoints are in different parts (i.e. $\max_{(i,j) \in E: i \in V_a, j \in V_b, a \neq b} w_{ij}$).

Give a randomized $\frac{k-1}{k}$-approximation algorithm for the MAX k-CUT problem.

5.2 Consider the following greedy algorithm for the maximum cut problem. We suppose the vertices are numbered $1, \ldots, n$. In the first iteration, the algorithm places vertex 1 in U. In the kth iteration of the algorithm, we will place vertex k in either U or in W. In order to decide which choice to make, we will look at all the edges F that have the vertex k as one endpoint and whose other endpoint is $1, \ldots, k-1$, so that $F = \{(j,k) \in E : 1 \leq j \leq k-1\}$. We choose to put vertex k in U or W depending on which of these two choices maximizes the number of edges of F being in the cut.

(a) Prove that this algorithm is a 1/2-approximation algorithm for the maximum cut problem.

(b) Prove that this algorithm is equivalent to the derandomization of the maximum cut algorithm of Section 5.1 via the method of conditional expectations.

5.3 In the *maximum directed cut problem* (sometimes called MAX DICUT) we are given as input a directed graph $G = (V, A)$. Each directed arc $(i,j) \in A$ has nonnegative weight $w_{ij} \geq 0$. The goal is to partition V into two sets U and $W = V - U$ so as to maximize the total weight of the arcs going from U to W (that is, arcs (i,j) with $i \in U$ and $j \in W$). Give a randomized $\frac{1}{4}$-approximation algorithm for this problem.

5.4 Consider the non-linear randomized rounding algorithm for MAX SAT as given in Section 5.6. Prove that using randomized rounding with the linear function $f(y_i) = \frac{1}{2}y_i + \frac{1}{4}$ also gives a $\frac{3}{4}$-approximation algorithm for MAX SAT.

5.5 Consider the non-linear randomized rounding algorithm for MAX SAT as given in Section 5.6. Prove that using randomized rounding with the piecewise linear function

$$f(y_i) = \begin{cases} \frac{3}{4}y_i + \frac{1}{4} & \text{for } 0 \leq y_i \leq \frac{1}{3} \\ 1/2 & \text{for } \frac{1}{3} \leq y_i \leq \frac{2}{3} \\ \frac{3}{4}y_i & \text{for } \frac{2}{3} \leq y_i \leq 1 \end{cases}$$

also gives a $\frac{3}{4}$-approximation algorithm for MAX SAT.

5.6 Consider again the maximum directed cut problem from Exercise 5.3.

(a) Show that the following integer program models the maximum directed cut problem:

$$\begin{array}{lrl} \text{maximize} & \displaystyle\sum_{(i,j)\in A} w_{ij} z_{ij} & \\ \text{subject to} & z_{ij} \leq x_i, & \forall (i,j) \in A, \\ & z_{ij} \leq 1 - x_j, & \forall (i,j) \in A, \\ & x_i \in \{0,1\}, & \forall i \in V, \\ & 0 \leq z_{ij} \leq 1, & \forall (i,j) \in A. \end{array}$$

(b) Consider a randomized rounding algorithm for the maximum directed cut problem that solves a linear programming relaxation of the integer program and puts vertex $i \in U$ with probability $1/4 + x_i/2$. Show that this gives a randomized 1/2-approximation algorithm for the maximum directed cut problem.

5.7 In this exercise, we consider how to derandomize the randomized rounding algorithm for the set cover problem given in Section 1.7. We would like to apply the method of conditional expectations, but we need to ensure that at the end of the process we obtain a valid set cover. Let X_j be a random variable indicating whether set S_j is included in the solution. Then if w_j is the weight of set S_j, let W be the weight of the set cover

(d) Prove the local ratio theorem.

(e) Explain how the set cover algorithm above can be analyzed in terms of the local ratio theorem to prove that it is an f-approximation algorithm.

Most of the algorithms of the chapter have local ratio variants.

7.4 In the 2-approximation algorithm for the generalized Steiner tree problem that we gave in Section 7.4, we first add certain edges, then remove unnecessary edges in the order opposite of the order in which they were added.

Prove that one can in fact remove unnecessary edges in *any* order and still obtain a 2-approximation algorithm for the problem. That is, we replace the edge removal steps in Algorithm 7.6 with a step that checks if there exists any edge e in F' such that $F' - e$ is feasible. If so, e is removed from F', and if not, F' is returned as the final solution. Prove that $\sum_{e \in F'} c_e \leq 2 \sum_S y_S$ for the dual y generated by the algorithm.

7.5 In the *minimum-cost branching problem* we are given a directed graph $G = (V, A)$, a root vertex $r \in V$, and weights $w_{ij} \geq 0$ for all $(i,j) \in A$. The goal of the problem is to find a minimum-cost set of arcs $F \subseteq A$ such that for every $v \in V$, there is exactly one directed path in F from r to v. Use the primal-dual method to give an optimal algorithm for this problem.

7.6 Recall that in our algorithms of Sections 4.4 and 5.7 for the prize-collecting Steiner tree problem, we used the following linear programming relaxation of the problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e x_e + \sum_{i \in V} \pi_i (1 - y_i) \\ \text{subject to} \quad & \sum_{e \in \delta(S)} x_e \geq y_i, && \forall i \in S, \forall S \subseteq V - r, S \neq \emptyset, \\ & y_r = 1, \\ & y_i \geq 0, && \forall i \in V, \\ & x_e \geq 0, && \forall e \in E. \end{aligned}$$

Given an optimal solution (x^*, y^*) to the linear program, we then selected a set of vertices U such that $U = \{i \in V : y_i^* \geq \alpha\}$ for some value of $\alpha > 0$.

Give a primal-dual algorithm that finds a Steiner tree T on the set of terminals U such that

$$\sum_{e \in T} c_e \leq \frac{2}{\alpha} \sum_{e \in E} c_e x_e^*.$$

(Hint: you should not need to design a new primal-dual algorithm.)

7.7 In the *k-path partition problem*, we are given a complete, undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ that obey the triangle inequality (that is, $c_{(u,v)} \leq c_{(u,w)} + c_{(w,v)}$ for all $u, v, w \in V$), and a parameter k such that $|V|$ is a multiple of k. The goal is to find a minimum-cost collection of paths of k vertices each such that each vertex is on exactly one path.

A related problem is that of partitioning a graph into 0(mod k)-trees. The input to this problem is the same as that above, except that the graph is not necessarily complete and edge costs do not necessarily obey the triangle inequality. The goal is to find a minimum-cost collection of trees such that each tree has 0(mod k) vertices, and each vertex is in exactly one tree.

(a) Given an α-approximation algorithm for the second problem, produce a $2\alpha\left(1-\frac{1}{k}\right)$-approximation algorithm for the first.

(b) Use the primal-dual method to give a 2-approximation algorithm for the second problem.

(c) Give a $4\left(1-\frac{1}{k}\right)$-approximation algorithm for the problem of partitioning a graph into *cycles* of length exactly k.

7.8 Show that the performance guarantee of the primal-dual algorithm for the uncapacitated facility location algorithm in Section 7.6 can be strengthened in the following way. Suppose that the algorithm opens the set T' of facilities and constructs the dual solution (v, w). Show that

$$\sum_{j\in D} \min_{i\in T'} c_{ij} + 3\sum_{i\in T'} f_i \leq 3\sum_{j\in D} v_j.$$

7.9 Show that for the k-median problem as defined in Section 7.7, the optimal solution can be found in polynomial time if the optimum cost $OPT_k = 0$.

7.10 By using the method of conditional expectations, show that the randomized algorithm for choosing k facilities in the k-median algorithm of Section 7.7 can be made deterministic.

Chapter Notes

The primal-dual method for approximation algorithms is a generalization of the primal-dual method used for linear programming and combinatorial optimization problems such as the shortest s-t path problem, the maximum flow problem, the assignment problem, the minimum-cost branching problem, and others. For an overview of the primal-dual method and its application to these problems, see Papadimitriou and Steiglitz [241]. Edmonds [96] gives the primal-dual algorithm for the minimum-cost branching problem in Exercise 7.5. The idea of Section 7.3 that the shortest s-t path problem can be solved by an algorithm that greedily increases dual variables is due to Hoffman [169]. Dijkstra's algorithm for the same problem is due, of course, to Dijkstra [89].

The first use of the primal-dual method for approximation algorithms is due to Bar-Yehuda and Even [35]; they gave the algorithm of Section 7.1 for the set cover problem. Work in primal-dual approximation algorithms was revived by work on the generalized Steiner tree problem of Section 7.4. The first 2-approximation algorithm for the generalized Steiner tree problem is due to Agrawal, Klein, and Ravi [4], and the algorithm presented in Section 7.4 is essentially that of [4]. The use of linear programming and LP duality in the algorithm was made explicit Goemans and Williamson [140], who extended the technique to other problems (such as the k-path partition problem of Exercise 7.7). The idea of depicting dual variables as moats is due to Jünger and Pulleyblank [182].

Several uses of the primal-dual method for approximation algorithms then followed. Bar-Yehuda, Geiger, Naor, and Roth [37] gave the feedback vertex set algorithm of Section 7.2, using Lemma 7.3, which is due to Erdős and Pósa [101]. Jain and Vazirani [179] developed the primal-dual algorithm for the uncapacitated facility location problem and the use of Lagrangean

For each $v \in V$ the sum telescopes to $\ln(V_d(v,\Delta))+\ln(V_d(v,\Delta/2))-\ln(V_d(v,1))-\ln(V_d(v,1/2))$, which can be bounded above by $2(\ln(V_d(v,\Delta)) - \ln(V_d(v,0)))$. Thus this sum is at most

$$32\sum_{v\in V}\ln\left(\frac{V_d(v,\Delta)}{V_d(v,0)}\right)g(v).$$

Then since $V_d(v,\Delta)$ is at most V^*, the entire volume, plus the extra V^*/n term, while $V_d(v,0) = V^*/n$,

$$32\sum_{v\in V}\ln\left(\frac{V_d(v,\Delta)}{V_d(v,0)}\right)g(v) \le 32\sum_{v\in V}\ln\left(\frac{V^*+V^*/n}{V^*/n}\right)g(v) = 32\ln(n+1)\sum_{v\in V}g(v).$$

Using the definition of $g(v)$, we have that

$$32\ln(n+1)\sum_{v\in V}g(v) = 32\ln(n+1)\sum_{v\in V}\left(\frac{V^*}{n}+\sum_{u\in V}c_{uv}d_{uv}\right) = 96\ln(n+1)\sum_{u,v\in V}c_{uv}d_{uv},$$

so that

$$\sum_{u,v\in V}c_{uv}T_{uv} \le O(\log n)\sum_{u,v\in V}c_{uv}d_{uv},$$

as desired. □

While we have gone to considerable lengths to give a deterministic algorithm to find a tree metric T such that $\sum_{u,v\in V}c_{uv}T_{uv} \le O(\log n)\sum_{u,v\in V}c_{uv}d_{uv}$, we can quite simply obtain a randomized algorithm that finds such a tree metric with high probability given a randomized algorithm for probabilistically approximating a metric by a tree metric with distortion $O(\log n)$. We give this as an exercise (Exercise 8.12). The reverse direction can also be shown; given any deterministic algorithm to find a tree metric T such that $\sum_{u,v\in V}c_{uv}T_{uv} \le O(\log n)\sum_{u,v\in V}c_{uv}d_{uv}$, we can obtain a randomized algorithm that can probabilistically approximate d by a tree metric with distortion $O(\log n)$. We give the latter problem as an exercise later on in the book, once we have a bit more experience with the ellipsoid method (Exercise 15.9).

Exercises

8.1 Prove that the analysis of the performance guarantee of the multiway cut algorithm of Section 8.2 can be improved to $\frac{3}{2}-\frac{1}{k}$.

8.2 Consider the following two permutations π_1 and π_2, where $\pi_1(1)=1,\pi_1(2)=2,\ldots,\pi_1(k)=k$, while $\pi_2(1)=k,\pi_2(2)=k-1,\ldots,\pi_2(k)=1$. Consider a modification of Algorithm 8.1 in which we do not choose a random permutation π, but rather choose $\pi=\pi_1$ with probability 1/2 and $\pi=\pi_2$ with probability 1/2. Show that the modified algorithm is still a $\frac{3}{2}$-approximation algorithm for the multiway cut problem.

8.3 In the *Steiner k-cut problem*, we are given an undirected graph $G=(V,E)$, costs $c_e \ge 0$ for all $e \in E$, a set of terminals $T \subseteq V$, and a positive integer $k \le |T|$. The goal of the problem is to partition the vertices into k sets $S_1,\ldots,S_k$ such that each set contains at least one terminal (that is, $S_i \cap T \ne \emptyset$ for $i=1,\ldots,k$) and to minimize the weight of the edges with endpoints in different parts. Given a partition $\mathcal{P}=\{S_1,\ldots,S_k\}$, let $c(\mathcal{P})$ be the total cost of the edges that have endpoints in different parts of the partition.