

Solution Sketch for Problem Set 2

1. (25 points) Synchronized duals for set cover

A primal-dual approach that we have studied is to start with a feasible dual solution and then raise a certain subset of the dual variables until some dual constraint became tight, at which point we set the associated primal variable appropriately. For the Steiner Forest problem in particular, we studied a scheme in which we raise a set of “active” dual variables simultaneously at a uniform rate until some edge became tight.

In this problem, we analyze the same approach for set cover. Consider the following algorithm. Recall that we have a primal variable x_S for each set S in the collection of sets and a dual variable y_e for each element of the universe.

- Set all y_e to 0. Initialize set cover F to \emptyset .
- Repeat the following steps until all elements are covered.
 - Raise the dual variable y_e for each uncovered element uniformly until some set S becomes “tight” (i.e., $\sum_{e \in S} y_e = c(S)$).
 - Add S to F and remove any sets all of whose elements have been covered. (If multiple sets become tight, let S denote an arbitrary one tight set.)
- Return F .

Show that the above algorithm has an approximation ratio of at most f , where f is the *maximum frequency*, the maximum number of sets that contain a given element. Show that for any value of the maximum frequency f , there exists an instance for which the algorithm’s approximation ratio is at least $f - \varepsilon$, where $\varepsilon > 0$ can be made arbitrarily small.

Answer: For each set S picked, c_S equals the sum of the y_e ’s. Thus, the cost of the set cover equals

$$\sum_S \sum_{e \in S} y_e.$$

The y_e ’s computed above do yield a feasible solution to the dual of cost $\sum_e y_e$. Since each e appears in the above displayed equation at most f times, we have an f -approximation.

For a lower bound on the approximation ratio, consider the following set cover instance. The universe is $U = \{a_1, a_2, \dots, a_f, v_1, v_2, \dots, v_m\}$ for $m = f/\delta$, where δ is specified later. The sets are U of cost $f + m$ and $S_i = \{a_i, v_1, \dots, v_m\}$ of cost $m - 1$. The main purpose of setting up the instance this way is to ensure that the above algorithm selects the sets S_i , for all i , while an optimal solution will select U .

The approximation ratio is $f(m - 1)/(f + m)$. By choosing δ sufficiently small (say $\leq \varepsilon/f$), the above ratio can be made larger than $f - \varepsilon$.

2. (25 points) Including tight edges in primal-dual algorithm for Steiner Forest

The Goemans-Williamson primal-dual algorithm we studied in class includes only one tight edge in any iteration. We discussed the primal-dual algorithm in which we include all tight edges in an iteration, and in the last step consider edges in arbitrary order, removing any edge that forms a cycle.

Show that if the order of edges in the last step is unspecified, then the above algorithm is $\Omega(n)$ -approximate, where n is the number of nodes.

(*Hint:* Devise an example in which we add light-weight edges followed by several heavy-weight edges (all in a single iteration), and then in the edge deletion step, we first remove the light-weight edges since they are part of some cycle.)

Answer: Consider a node s connected to $m = n - 2$ nodes v_1 through v_m , each with weight 1, and a node t connected to v_1 through v_m each with weight W . Suppose we are seeking a minimum spanning tree.

In the first iteration, all the weight-1 edges are selected. In the second, all the weight- W edges are selected. In the deletion step, if all the heavy-weight edges are considered first, then the final spanning tree is of cost $mW + 1$. An optimal MST includes all weight-1 edges and one weight- W edge for a total cost of $W + m$. The ratio of the two is $\Omega(n)$, if $W = \Omega(n)$.

3. (25 points) Initial solution for k -median local search

The running time of the local search algorithm for k -median is simply the number of iterations times the time taken for each local search iteration. Since the latter is fixed, it makes sense to keep the number of iterations small. One way toward this is to start with a reasonable initial solution. Assume for simplicity that all points have unit demand and a facility can be potentially opened at any point. Consider the following initial solution.

- $S \leftarrow \emptyset$.
- For i from 1 to k :
 - Let u be a point whose distance to the current set is largest; that is, $d(u, S) \geq d(v, S)$ for all v , where $d(v, S)$ is defined as $\min_{w \in S} d(v, w)$.
 - Add u to S .

- (a) Let S be the set of points returned by the above algorithm and let T be an arbitrary set of k points. Prove that for

$$\max_v d(v, S) \leq 2 \cdot \max_v d(v, T).$$

(*Hint:* One approach is to use induction on k , the number of points in S and T . The base case is easy. For the induction step, consider two cases. One case is when there exists a point t in T for which there are at least two points in S , whose closest point in T is t . The other case is when for every point t in T , there is exactly one point in S whose closest point in T is t .)

Answer: When I starting writing the solution down, I realized that an induction proof seems more complicated than necessary. So here is a more direct proof.

We map every point in S to its closest point in T . Consider any point x . Let t_x be the closest point in T . If there is no point in S that is mapped to t_x , then it must be the case that there exists a point t in T for which there are at least two points in S , say s_1 and s_2 , whose closest point in T is the same, say t' . Without loss of generality, let us assume s_1 was added first in the above algorithm. When s_2 was added, the distance of x to its nearest point in the set S at that time, say S' , was at most the distance of s_2 to S' . So,

$$d(x, S) \leq d(x, S') \leq d(s_2, S') \leq d(s_2, s_1) \leq d(s_2, t') + d(s_1, t') \leq 2 \cdot \max_v d(v, T).$$

The second case is when there is a point in S , say s_x , whose closest point in T is t_x . Then, we have:

$$d(x, s_x) \leq d(x, t_x) + d(s_x, t_x) \leq 2 \cdot \max_v d(v, T),$$

completing the proof.

(Remark: The above algorithm is called the farthest-point heuristic and is due to T. Gonzalez. See his paper on clustering in *Theoretical Computer Science*, 1985. And the problem of selecting k medians so as to minimize the maximum distance to the closest median is called the k -center problem.)

- (b) Using part (a), prove that the above algorithm yields a k -median solution that is $O(n)$ -approximate.

Answer: Let S be the k -median solution. The cost of S is at most n times the maximum distance of a client to a median, which by (a), is at most twice the maximum distance of a client in the optimal k -median solution. A bound of $2n$ on the approximation ratio follows.

Remark: While this is not a good approximation ratio, the fact that it is independent of the value of the distances in the metric makes it useful to show that the running time of local search, used with this procedure, is strongly polynomial time.)

4. (25 points) LPs for multicommodity flow.

Consider the demands version of the multicommodity flow problem. We are given a graph $G = (V, E)$ with capacity on edges given by $c_e, e \in E$, a set of k source-sink pairs $(s_i, t_i), 1 \leq i \leq k$ with demands given by $d_i, 1 \leq i \leq k$. The goal is to determine the largest f such that $f \cdot d_i$ flow of the i th commodity can be sent from source s_i to sink t_i , for $1 \leq i \leq k$, subject to the flow conservation and capacity constraints.

In our class (when we covered the Leighton-Rao result), we wrote the following dual LP for the path-flow-based LP.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e d_e \\ \sum_{e \in p_i^j} d_e \quad & \geq \quad l_i \quad \forall s_i - t_i \text{ path } p_i^j \\ \sum_{1 \leq i \leq k} l_i \quad & \geq \quad 1 \end{aligned} \tag{1}$$

- (a) Write down an alternative linear program for the demands multicommodity flow problem based on edge flows. In particular, have a variable $x_{(u,v)}^i$ for each edge (u, v) and commodity i that represents the flow of commodity i from u to v . Complete the LP by capturing the objective function, the flow conservation constraints, and capacity constraints in terms of these “edge flow” variables.

Answer: The objective is to maximize the fraction f of the demand that can be satisfied for each commodity. For each commodity i , we have flow conservation constraints at each node (except the source and sink of i), as in a regular network flow. For each directed edge (u, v) , we have a capacity constraint

$$\sum_i x_{(u,v)}^i \leq c_{(u,v)}.$$

Finally, for each commodity i , we have the constraint that the total flow of i leaving the sink s_i is at least $f \cdot d_i$.

This completes the LP description.

- (b) Write a dual for the linear program from part (a). Compare this dual with the LP given by Equation 1.

Answer: In the dual, we have a variable d_{uv} for each edge (u, v) , and a “length” variable ℓ_u^i for each commodity i and node $u \neq t_i$. For convenience, we also add the variable $\ell_{t_i}^i$, which is set to 0. The dual is

$$\begin{aligned} \max \quad & \sum_{(u,v) \in E} c_{(u,v)} d_{(u,v)} \\ & \sum_i \ell_{s_i}^i d_i \geq 1 \\ \ell_u^i \leq & \ell_v^i + d_{(u,v)} \quad \forall i, \forall v \neq t_i \\ & d_{(u,v)} \geq 0 \quad \forall (u, v) \end{aligned}$$

The dual above is equivalent to the LP in Equation 1. The variables $d_{(u,v)}$ are the edge lengths in each LP, and l_i of the first LP is the same as $\ell_{s_i}^i$ of the second LP; in each case, it represents the distance between s_i and t_i according to the edge distance labels.

- (c) Using part (b) or otherwise (e.g., using the separation oracle method), show that LP 1 can be solved in polynomial time.

Answer: One can solve LP 1 by solving the LP of part (b), which is of polynomial size and can be solved using any polynomial LP algorithm.

5. (25 points) Unsplittable multicommodity flow

In the multicommodity flow problem we defined in class, we assumed that a flow from a source to a sink can be split among multiple paths. In some applications, especially in the networking domain, one may demand flow of a single commodity to be sent along a single path. We call this the unsplittable multicommodity flow problem.

- (a) Show that the unsplittable multicommodity flow problem is NP-hard, even with two commodities.

(Hint: Reduce from the partition problem.)

Answer: To prove the above claim for two commodities (in directed graphs), I do not think reducing from partition would be easy. There is a reduction from SAT, due to Fortune and Wylie.

Here, we give the reduction from partition with larger number of commodities. Suppose the partition instance has numbers a_1 , through a_n adding up to $2M$. We have n sources s_1 through s_n , all located at node A , and n corresponding sinks t_1 through t_n , all located at node B . There are two edges from A to B , each of capacity M (if you wish, you can add intermediate points in those edges). The demand for commodity i is a_i .

It is easy to see that the partition instance can be partitioned iff the above multicommodity flow problem admits an unsplittable flow for each commodity of value a_i .

Consider the special case of the demands version of the unsplittable multicommodity flow problem in which the demand of each commodity as well as the capacity of each edge is 1. We study the following randomized rounding algorithm for unsplittable multicommodity flow in this case.

1. Solve the LP relaxation of the problem (which removes the unsplittability constraint).
 2. Decompose the flow of each commodity i into a set P_i of at most m flow paths, where m is the number of edges. For each commodity i , select a path at random from P_i with probability equal to the flow of i along the path. Send all the flow for commodity i (as determined by the LP) along this path. Call the resulting flow f .
 3. Return the flow obtained by scaling f down by the largest amount of flow going across any edge.
- (b) Prove that after step 2 of the above algorithm, the expected flow along any edge is at most 1.

Answer: One point that is confusing in the way the above algorithm is stated is whether we send a unit flow in step 2. I should have clearly stated an implicit assumption that the LP relaxation yields a unit flow for every commodity. If not, we can appropriately scale the demands and then apply the algorithm.

In the remainder, we assume that the LP solution yields a unit flow for every commodity. So in step 2, whenever a path is selected, a unit flow is sent along the path. The expected flow along any edge is simply the sum of the path flows, over all commodities – since each path is selected with probability equal to its flow. Since the capacity of each edge is 1, the desired result follows.

- (c) Prove that after step 2, the flow along any edge is at most $O(\log n)$ with probability at least $1 - 1/n^3$.

Hint: Use Chernoff bounds. See the theorem on relative error in the following wikipedia entry.

http://en.wikipedia.org/wiki/Chernoff_bounds

Answer: The flow along any edge is the sum of independent random variables X_i , one for each commodity. (One could instead have a flow variable for each path-commodity pair. In this case, the variables will not be independent, so directly applying Chernoff-type bounds is

difficult.) Each variable is a 0 – 1 variable, and the sum of the expectations of these variables is at most 1, by part (b).

By applying a Chernoff-type bound with error bound $\delta = \Theta(\log n)$, we obtain the probability that the flow along any edge exceeds $\Omega(\log n)$ is at most $(e/\Omega(\log n))^{\Omega(\log n)}$, which is at most $1/n^3$, by choosing constants appropriately in the asymptotic notation.

- (d) Conclude that the above algorithm yields an unsplittable multicommodity flow whose value is at most an $O(\log n)$ factor less than that of an optimal unsplittable multicommodity flow, with probability at least $1 - 1/n$.

Answer: Since there are at most n^2 edges, a simple union bound shows that the probability that any of edge capacity is exceeded by $\Omega(\log n)$ factor is at most $1/n$. Thus, if we scale down the flows by $O(\log n)$ factor, we obtain an $O(\log n)$ -approximation, with probability at least $1 - 1/n$.