Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
David Karger

Handout 20
November 7, 2012

Problem Set 9 Solutions

Problem 1. For the i-th request pair, let x_i and y_i denote the indicator variable that we use the clockwise path and anticlockwise path for the connection. Consider the following LP relaxation of the original problem:

$$\begin{aligned}
\min \quad & z \\
x_i + y_i \quad & \geq 1 \forall i \\
z \quad & \geq \sum_{\text{the path for } x_i \text{ contains } e} x_i + \sum_{\text{the path for } y_i \text{ contains } e} y_i \forall e \\
x_i \quad & \geq 0 \forall i \\
y_i \quad & \geq 0 \forall i
\end{aligned}$$

We round any $x_i, y_i \geq 1/2$ to 1. Observe that for each request pair, at least one of $x_i \geq 1/2$ and $y_i \geq 1/2$ must hold, and we choose the path whose indicator variable is rounded up to 1 (if both paths are rounded up to 1, choose one arbitrarily). To maintain the feasibility of the second set of constraints, we can simply double z, which corresponds to the maximum load. Thus this algorithm is a 2-approximation.

Problem 2. **(a)** An integer variable x_{jt}, for $j = 1, \ldots, n$, where n is the number of jobs, and $t = 1, \ldots, \sum_j p_j$, is an indicator that job j completed at time t. Obviously,

$$\forall j \forall t, \qquad 0 \leq x_{jt} \leq 1.$$

We enforce that every job completes:

$$\forall j \qquad \sum_t x_{jt} = 1,$$

that a job is not processed before its predecessors:

$$\forall j \forall k \in A(j) \forall t \qquad \sum_{i=1}^{t-p_k} x_{kt} \geq \sum_{i=1}^{t} x_{jt},$$

and that the total processing time of jobs completed before time t is at most t:

$$\forall t \qquad \sum_{i=1}^{t} \sum_j p_j x_{ji} \leq t.$$

We only need to specify the goal function. It is

$$\min \sum_t \sum_j w_j t x_{jt}.$$

(b) For a job j and its halfway point h_j,

$$\overline{C}_j = \sum_t t x_{jt} \geq \sum_{t \geq h_j} t x_{jt} = h_j \sum_{t \geq h_j} x_{jt} \geq \frac{1}{2} h_j$$

Voilà!

(c) The set of constraints that worked for the integer linear program, and expressed that a job j cannot be processed before its predecessor k, now says that a half of j must be processed after a half of k. This implies that no job runs before its predecessor.

(d) Note that at the halfway point of job j at least half of each of the jobs preceding j in the order has been completed. This implies that a half of the sum of their (including j) processing times is not greater than h_j, and therefore the completion time of j is at most $2h_j$. By part (b) we know that in turn their completion time is not greater that $4\overline{C}_j$.

(e) Solving the linear program, we minimize function

$$\sum_t \sum_j w_j t x_{jt} = \sum_j w_j \sum_t t x_{jt} = \sum_j w_j \overline{C}_j.$$

The minimum we achieve is no worse that the minimum for the integer linear program, and because our completion times are not worse than four times the average completion times in the optimal solution, we achieve a constant-factor approximiation for $1 \mid prec \mid \sum C_j$.

Problem 3. **(a)** If we break each vertex into two, an "entry node" and an "exit node", the problem of finding a cycle cover is simply a bipartite matching between entry nodes and exit nodes. To find the minimum cycle cover, simply solve the min-cost maximum matching algorithm by using the min-cost max-flow algorithm as shown in class.

(b) Each cycle will have at least two nodes, and by our bipartite matching, the cycles are vertex-disjoint, i.e. there are no self-loops. Thus if we choose one representative node from each cycle, there can be at most half of the total number of nodes in the graph. In addition, the optimum tour traversing only these representative nodes must cost less than or equal to the original optimum because the edges all satisfy the triangle inequality. Requiring that our tour visit other nodes can only increase the cost.

(c) If we repeat this finding and selecting representatives from minimum cycle covers, we at least halve the number of nodes in each step until we get down to one node. This clearly takes $O(\log n)$ steps. When we unravel the selections we've done, we essentially take the cycle represented by each represented node. This is clearly a tour of the whole graph because the cycle covers in each stage must cover every vertex and the final tour we take starts and ends at the same node. Since the cost of the minimum cycle cover of a set of nodes is a lower bound on the cost of the optimum tour over the same nodes, which can be at most OPT in any of the steps we took, we have a total cost of at most $O(\log n)$OPT.

Problem 4. **(a)** To get a 9-approximation, we double the distance we're testing at each iteration. In other words, we go one unit of distance up the river, then search 2 units downriver from our starting point, then 4 units upriver, and so on. On the 0-th step, we travel 1 unit, and on each step $i > 0$, we travel a distance of $2^i + 2^{i-1}$. In the worst case, we take a step in some direction and fall just short of finding the bridge. We turn around, go a bit less than $3d$ away to check the other side, and then come all the way back to find the bridge for a total extra $6d$ cost. We took about $\lg d$ steps to get to the place right before the bridge, and that corresponds to a distance of

$$\begin{aligned} 1 + \sum_{i=1}^{\lg d} (2^i + 2^{i-1}) &= 2 \sum_{i=0}^{\lg d} 2^i - d \\ &= 2(2d - 1) - d \\ &= 3d - 1 \end{aligned}$$

Adding in the $6d$ in the worst case gives us $9d$ for a 9-approximation.

(b) With randomization, we can improve our result from our deterministic algorithm. If we pick uniformly at random whether to start looking upstream or downstream, then half the time, we'll still pay the $6d$ extra cost, and the other half the time, we'll pay something less. Given the same worst-case situation described in the previous part, with probability $1/2$, the last step that reaches a distance less than d from the starting point will be in the opposite direction from the bridge. In this case, it only requires at most another $2d$ to get to the correct location of the bridge on the other side of the starting point. This best case thus has a competitive ratio of $3 + 2 = 5$. Since these best and worst cases each happen with probability $1/2$, the overall competitive ratio is 7.

Problem 5. To implement MTF' (MTF-every-other), we keep a mark bit for each element. If we access a marked element, we do not move it, and unmark it. If we access an unmarked element, we move it to the front of the list and mark it. Initially, all elements are unmarked.

Consider the lists l for MTF and l' for OPT. Write $i <_l j$ if i precedes j in l, and similarly $i <_{l'} j$. We define two potential functions as follows.

$$\begin{aligned}\phi &= |\{(i,j) : i <_l j, i >_{l'} j, j \text{ unmarked}\}| \\ \phi' &= |\{(i,j) : i <_l j, i >_{l'} j, j \text{ marked}\}|\end{aligned}$$

The potential function we consider is

$$\Phi = \phi + 2\phi'$$

We will show that for any request at time t we have the following:

$$C_{MTF'} + \Phi_t - \Phi_{t-1} \leq 3C_{OPT}$$

where $C_{MTF'}$ and C_{OPT} are the real costs incurred through accesses. Note that $\Phi_0 = 0$, and $\Phi_i \geq 0$, so that if we sum over any n requests, we have that:

$$C_{MTF'} + \Phi_n - \Phi_0 \leq 3C_{OPT}$$

Since $\Phi_n - \Phi_0 \geq 0$, we have our desired result.

Consider a request for x, and let k be the index of x in l. We define the following two sets:

$$\begin{aligned}F &= \{i : i <_l x, i <_{l'} x\} \\ B &= \{i : i <_l x, i >_{l'} x\}\end{aligned}$$

and let $f = |F|, b = |B|$. That, is f is the number of elements ahead of x in l that are also ahead of x in l', and b is the number of elements ahead of x in l but behind x in l'. Clearly, we have $k = f + b + 1$.

If x was unmarked, then x is moved to the front of the list, and is marked. Note that every item in B is no longer an inversion. Additionally, each item in B previously contributed to ϕ, since x was marked. Therefore, ϕ decreases by b. Now, ϕ' increases by at most f. Each element of F is now an inversion; in the worst case, each element of F was marked. Therefore, we have that

$$\Delta\Phi \leq 2f - b$$

Now, if x was marked, then x is not moved and is unmarked. Now, ϕ increases by b by definition, and ϕ' decreases by b by definition. Therefore, in this case, we also have that:

$$\Delta\Phi = -b \leq 2f - b$$

Therefore, the cost of MTF' plus the change of potential is:

$$C_{MTF'} + \Delta\Phi \leq k + 2f - b = f + b + 1 + 2f - b \leq 3f + 1$$

Now let us examine the cost of OPT on an access to x. The real cost of accessing x must be at least f. Therefore, we have that

$$C_{MTF'} + \Delta\Phi \leq 3C_{OPT}$$

Problem 6. **(a)** Let $S = \{v_1, v_2 \dots v_k\} \subseteq V$ be an independent set of size k in G. For a vertex $v_i \in S$, define

$$f(v_i) = \cup_{j=1}^{k} \{v_i \in G_{v_k}\}$$

and consider the following set of vertices in G':

$$S' = \cup_{i=1}^{k} f(v_i).$$

Clearly, $|S'| = k^2$. Furthermore, S' is an independent set in G'; any two vertices in S' that exist in the same subgraph G_{v_i} clearly do not have an edge between them. For two vertices in S' that exist in different subgraphs $G_{v_i} \neq G_{v_j}$, we have that there is no edge, because no edge connects v_i and v_j in G, so there are no edges between vertices in G_{v_i} and G_{v_j}.

(b) Let S be the independent set of size s in the graph G'. Consider the subgraphs $G_{v_1} \dots G_{v_k}$, and let S' be the set of vertices whose subgraphs contain at least one vertex in S. Note that S' forms an independent set in G, since if no edges exist between G_{v_i} and G_{v_j}, then there is no edge between v_i and v_j in the original graph G.

If $|S'| \geq \sqrt{s}$, we are done. Otherwise, there exists one subgraph G_{v_i} with at least $s/\sqrt{s}$ vertices; in this case, we just take all the vertices in S' in this subgraph as an independent set.

(c) Suppose there exists an α approximation scheme for a fixed α. Given a graph, we can construct the product graph G', and find an independent set of size $\alpha \cdot \mathrm{OPT}_{G'}$ in G'. Using (b), we can construct an independent set of size $\sqrt{\alpha \cdot \mathrm{OPT}_{G'}}$. However, by part (a) and (b), we know that $\mathrm{OPT}_{G'} = \mathrm{OPT}_G^2$, so that we have constructed an independent set in G of size $\sqrt{\alpha} \cdot \mathrm{OPT}_G$.

By constructing a product graph G^t iteratively $t = \frac{\log \alpha}{\log(1-\epsilon)}$ times, we could find a maximum independent set in G of size $(1-\epsilon)\mathrm{OPT}$, i.e. a polynomial time approximation scheme. The argument follows from the above paragraph.