CME 305: Discrete Mathematics and Algorithms

Instructor: Professor Amin Saberi (saberi@stanford.edu)

Midterm – 02/28/10

Problem 1. Show that a graph has a unique minimum spanning tree if, for every cut of the graph, the edge with the smallest cost across that cut is unique. Show that the converse is not true by giving a counterexample.

Solution: Suppose MST is not unique, i.e., there exist T_1 and T_2 where both of them are MST and they are not identical. Suppose $e_1 \in T_1$ but $e_1 \notin T_2$, if we remove e_1 from T_1, then we will have two trees with vertex sets V_1 and V_2. By problem 1 of HW #2, we know that e_1 is a minimum cost edge in the cut between V_1 and V_2. Now consider T_2, again problem 1 of HW #2, we know that T_2 contains an edge e_2 that is a minimum cost edge of the cut between V_1 and V_2. However, since $e_2 \neq e_1$ we must have: $c(e_1) = c(e_2)$ which is contradicting with the assumption that for every cut of the graph, the edge with the smallest cost across that cut is unique.

Counter-example for the converse: suppose the graph is a tree with 3 nodes, where both edges have cost 1. Let v be the node with degree two. Clearly the only spanning tree is the graph itself, but the cut between v and the two leaves contains two cost 1 edges, so the minimum cost edge is not unique across this cut.

Problem 2. The *edge connectivity* $\lambda(G)$ of an undirected graph $G(V, E)$ is defined as the cardinality of a minimum set of edges $S \subseteq E$, whose removal disconnects G. Give a polynomial-time algorithm for computing $\lambda(G)$.

Solution: There are $k = n(n-1)/2$ distinct pairs of vertices in G. We solve k maximum flow problems, one for each pair, and pick the one with the smallest maximum flow. Each flow problem for vertex pair $\{i, j\}$ is formulated as follows. For each edge in G, replace it with two directed edges i.e. for each $\{i, j\}$ define directed edges (i, j) and (j, i). Define a flow network N by introducing a source s and sink t to the modified graph. Connect s to vertex i with a directed edge (s, i) and j to t with directed edge (j, t). Each max-flow in each of the k networks then corresponds to the min cut in N by the max-flow min-cut theorem. We pick the minimum of these k values. This min-cut defines the set S in a straightforward way since we have for any cut $(S, \bar{S})$ in G there is a vertex pair $\{i, j\}$ for which we've solved a max-flow problem and thus found a min-cut. Our algorithm takes polynomial time since we max-flow can be solved in polynomial time and we solve a polynomial number k of such problems.

Problem 3. Recall the job scheduling problem. We have m machines and n jobs such that any machine j takes time t_i to process job i. Let A_j be the set of jobs assigned to machine j. In that case, $T_j = \sum_{i \in A_j} t_i$ will be the *load* of machine j. Our goal is to find an assignment of jobs to machines that would minimize $\max_j T_j$. Denote this minimum value OPT.

Consider a slightly modified greedy approach to the one showed in class. First, we sort the jobs so that $t_1 \geq t_2 \geq \cdots \geq t_n$. Then, we assign them iteratively to the machines, every time to the machine with the smallest load.

Show that the approximation factor of this algorithm is at most 3/2.

Hint: Note that if $n > m$ then $t_{m+1} \leq OPT/2$. Why is this true?

Extra Credit: Prove that the approximation factor of the algorithm is actually equal to 4/3.

Solution: If there are at most m jobs, the scheduling is optimal since we put each job on its own machine. If there are more than m jobs by the pigeonhole principle, at least one processor must get 2 of the first $m+1$ jobs. Each of these jobs is at least as big as t_{m+1}. Thus, as in the hint $OPT \geq 2t_{m+1}$.

Consider machine j assigned maximum load T where $j > m$ since otherwise we are done. Let i be the last job assigned to j. By the hint and since the loads are sorted $t_i \leq t_{m+1} \leq OPT/2$. Then, following the notes in class

$$T = (T - t_i) + t_i \leq \frac{1}{m}\sum_{j=1}^{m} T_j + t_i \leq OPT + OPT/2 = \frac{3}{2}OPT$$

Extra Credit: First, lets assume that $n > 2m$. Applying the same reasoning as above we have $OPT \geq 3\, t_{2m+1} \geq 3\, t_i$ for $i > 2m+1$ and as before we have

$$T = (T - t_i) + t_i \leq \frac{1}{m}\sum_{j=1}^{m} T_j + t_i \leq OPT + OPT/3 = \frac{4}{3}OPT$$

where T is the makespan produced by the algorithm.

Now, we need to show that for $n \leq 2m$, the algorithm produces a makespan with value $T = OPT$. Consider the following properties of the optimal assignment for jobs:

1. Each job $i \leq m$ is assigned its own machine. Otherwise, any two such jobs on a machine would produce a load greater than T. Thus, w.l.o.g. we can assign each such job i to machine $j = i$ in the optimal assignment.

2. By the same reasoning, the next $k \leq m$ jobs have to be placed on machines $m, m-1, \ldots, m-k+1$. Otherwise, we again would exceed T.

Finally, the optimal assignment of the last k jobs has to follow the algorithm placement since any interchange of the last job with any other increases the makespan. Thus, the algorithm reproduces the optimal makespan for $n \leq 2m$.