15-854 Approximation and Online Algorithms

Homework # 2 **due: February 16, 2000**

Groundrules: Same as hwk1.

Exercises:

Given a weighted undirected graph G, a *traveling salesman tour* for G is the shortest tour that starts at some node, visits all the vertices of G, and then returns to the start. We will allow the tour to visit vertices multiple times (so, our goal is the shortest cycle, not the shortest simple cycle). This version of the TSP that allows vertices to be visited multiple times is sometimes called the *metric* TSP problem, because we can think of there being an implicit complete graph H defined over the nodes of G, where the length of edge (u, v) in H is the length of the shortest path between u and v in G. (By construction, edge lengths in H satisfy the triangle inequality, so H is a metric.)

1. Briefly (at most a few sentences): prove that we can get a factor of 2 approximation to the TSP by finding a minimum spanning tree T for H and then performing a depth-first traversal of T.

2. The minimum spanning tree T (in fact any tree) must have an even number of nodes of odd degree (only considering the edges in the tree). Why?

3. Let M be a minimum-cost matching (in H) between the nodes of odd degree in T. I.e., if there are $2k$ nodes of odd degree in T, then M will consist of k edges no two of which share an endpoint. Prove that the total length of edges in M is at most one-half the length of the optimal TSP tour.

4. We have now proven that we can get a 1.5 approximation to the TSP if we can take an Eulerian tour on the union of T and M (a tour that traverses each edge in $T \cup M$ exactly once). Prove that such a tour exists and can be found efficiently.

The above algorithm is due to Christofides [1976]. Extra credit and PhD thesis: Find an algorithm that achieves a 1.49 approximation ratio.

Problems:

It is often nice if a database can give a user an estimate of the amount of time a search is going to take, before embarking on it. Below, we will work through a fast approximation algorithm for a problem based on this issue: given a directed graph G, determine the number of nodes reachable from each vertex in G (think of G as a database where a user queries some vertex v, and then the search engine performs some operation on all nodes reachable from v.) The best exact algorithms for this problem take time $O(\min(mn, n^{2.38}))$. We will work through a randomized approximation algorithm that takes time only $O(m \log n)$. This is a result due to Edith Cohen.

The basic idea of the algorithm is as follows. We give a random ranking to the nodes of G (that is, we label the nodes according to a random permutation of $\{1, \ldots, n\}$). Then, for each vertex v in G, we find the node of *lowest rank* reachable from v. Let's think about what we expect this rank to be: if v can reach only one node (i.e., itself), then the expected value of this rank is $(n + 1)/2$. If it can reach two nodes, then the expected value of the smaller of the two ranks is $(n+1)/3$. More generally, say v can reach R_v nodes, then the expected value of the smallest rank is $(n+1)/(R_v+1)$. So, this suggests that we can use these ranks to guess at the R_v's. If we want better guesses, we run this procedure several times and combine the information received. For technical reasons, we will deviate from this outline slightly. Specifically,

Here is the algorithm:

1. We give each vertex in G, independently, a random number in $[0,1]$. We then calculate for each vertex v, the *minimum* number m_v reachable from v.
2. We repeat Step 1 for t iterations. Let's say that for a given vertex v, this results in $m_{v,1}, m_{v,2}, \ldots, m_{v,t}$ as the t minima. Let $\tilde{m}_v$ be the median element in this list. That is, half of the m_i are larger than $\tilde{m}_v$ and half of the m_i are smaller.
3. For each vertex v, output $\tilde{r}_v = 1/\tilde{m}_v$ as our estimate of the number of vertices reachable from v. (Actually, a better estimate is $1/\tilde{m}_v - 1$, but let's just say $1/\tilde{m}_v$ for simplicity.)

5. Prove that if we pick a random subset of k elements from $\{1, 2, \ldots, n\}$, then the expected value of the smallest of these is $(n+1)/(k+1)$. (This fact was used in the intuition above, but won't acutally be needed in the analysis of the algorithm).

6. Describe a deterministic $O(m+n)$ time algorithm that given a directed graph with n nodes and m edges, will find, for every vertex, the node of least index reachable from that vertex. (This allows us to perform Step 1 efficiently, since as we pick the random numbers we can also sort the nodes and give them indices according to their ranking.)

7. Prove there exist constants c_1, c_2 such that for any vertex v, with probability $1 - e^{-\Omega(t)}$,

$$R_v/c_1 < \tilde{r}_v < c_2 R_v.$$

This means that after not too many iterations, with high probability *all* our estimates are within a constant factor of their true values.[1]

Hint: For the lower bound on $\tilde{r}_v$, think of each iteration of Step 1 as flipping a coin, with the event that $m_v > c_1/R_v$ as a "heads". Calculate a value of c_1 such that the probability of heads is small enough that it is unlikely we would have seen more than $t/2$ heads.

Extra Credit. Prove that with high probability we will get estimates that are off by only $(1+\epsilon)$, for an appropriate value of t. To do this, you may need to alter the algorithm slightly.

[1] If you want, you are free to modify the algorithm to prove this bound, though you should not need to.