CS270 Combinatorial Algorithms & Data Structures **Spring 2003**

Lecture 19: 4.1.03

Lecturer: Satish Rao *Scribes: Kevin Lacker and Bill Kramer*

19.1 Approximation Algorithms

Approximation algorithms are used when we know a problem is hard. It is a way to consider optimization versions of function problems. For instance, given a 3-CNF formula, finding a maximal truth assignment is essentialy a MAX3SAT. Approximation algorithms return solutions that are close to the optimal in some way. So, if $OPT(x)$ is the optimal solution for problem x, and $A(x)$ is our approximate solution, one can consider the approximation ratio of algorithm A as

$$\max_{|x|=n} \frac{A(x)}{OPT(x)}$$

19.2 Vertex Cover

Another example of this is vertex cover. The problem can be posed as follows: Given a graph $G = (V, E)$, find the smallest C that is a subset of V such that, for all (u, v) that exist in E, at least one of u and v is an element of C. In other words, find the smallest $C \subseteq V$ such that $\forall(u, v) \in E$ either u $\in$ C or v $\in$ C.

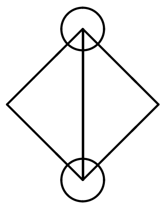

Figure 19.1: Example of the smallest vertex cover in a graph with 4 vertices as shown. The minimum vertex cover is 2 in this example, as indicated by the circles.

Start with an algorithm that at least finds a vertex cover but may not find the smallest. It may just be reasonable to try a greedy algorithm and see what happens. A simple algorithm for such a cover is:

```
C = 0; set C to the empty set
M = 0; set M to the empty set
For edge (u,v) that exists in E
  If u does not exist in C and v does not exist in C ; edge is not covered
     Put u and v into C ; cover the edge - twice for simplicity
     Put (u,v) into M
```

When the algorithm finishes, C is a vertex cover, and M is a set of edges for which both endpoints are in C.

We cannot prove this gets the smallest cover, but what can be proven about the algorithm?

Theorem 19.1 *If C^* is the optimal cover of G, then the minimal approximate cover C exists such that $|C| \leq 2 \times C^*$.*

This theorem states that no matter what the minimal cover, $C*$, of the graph is, the result of our algorithm is not more than a factor of 2 from the optimal. It is noted that the factor could depend on N and not be constant but not further explored in class.

Proof:

Look at C^* which is the optimal solution. C^* has to contain an endpoint of each edge in M. In addition, by the construction of M we know that the endpoints of the edges of M are precisely the set C.

Thus, $|C^*| \geq |M| \Longrightarrow |C| \leq 2 \times |M| < 2 \times |C^*|$

There is an open question. Is it possible to do better than a factor of two? Hastead used the PCP Theorem to prove doing better than 7/6 of optimal is an NP-complete problem. To understand PCP, think of NP-complete problems as having solution for which a proof can be written down. PCP (Probabilistically Checkable Proof) means that we can write down a proof, and then verify it, with good probability, just by spot checking a few bits of the proof.

19.3 Traveling Salesman Problem - TSP.

The basic problem is given a weighed graph $G = (V, E)$, find the shortest Hamiltonian tour. A Hamiltonian tour is a path that visits each city exactly once and then returns to the city of origination. This is often explained as planning the best route for a sales representative who has to travel to N given cities where each pair of cities has a specified distance between them. Another way to say this is given an undirected graph $G = (V, E)$, find a tour of the minimum length. Assume the triangle inequality holds so the $distance(x, z) \leq distance(x, y) + distance(y, z)$.

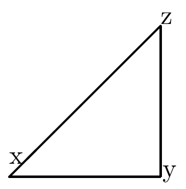

Figure 19.2: An example of the triangle inequality so $distance(x, z) \leq distance(x, y) + distance(y, z)$.

The concept is to:

- Compute a lower bound,
- Construct a solution,
- Find an upper bound

Theorem 19.2 *There is a 2-approximate algorithm for MIN-TSP.*

Proof:

Let T be an optimal tour. First, compute a minimal spanning tree (MST), M, for our graph G. Construct a tour T^* by traversing the MST twice. We can short-circuit any edges we have already crossed over. (The triangle inequality guarantees that short-circuiting cannot lengthen the tour).

$|T| \geq |M|$ since any tour must span G by definition. Also $|T^*| = 2 \times |M|$. So, $|T^*| = 2 \times |M| \geq |T| \geq |M|$.

This means that T^* is 2-approximate to T.

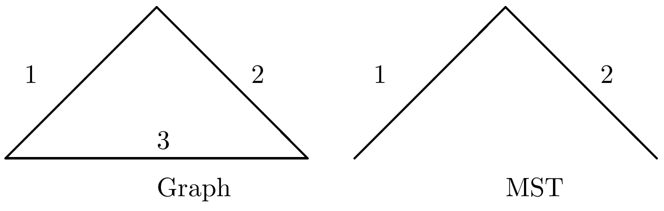

Figure 19.3: Example of a Graph $G = (V, E)$ and its MST

The tour can be viewed on the spanning tree.

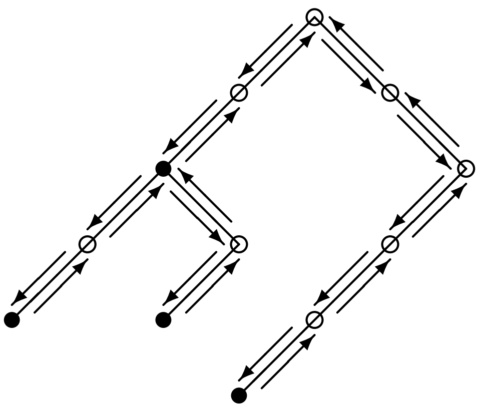

Figure 19.4: The MST traversal which starts at the root and visits every node finally returning the root. The arrows point the order of tranversing the tree for a depth first search. It is clear that each edge is traversed two times. The solid colored vertices are those with odd order, the open vertices are even order.

The length of a tour is the cost of the tour. In an MST, each edge is used twice, once in each direction. Tarjan (1976) conjectured that an MST is the best possible. Ironically, Christofides (1976) found a 3/2-approximation of TSP.

Look at the odd degree vertices in an MST. A tour is best if there is a tree plus one edge, with each vertex being connected with order 2. If we connect the odd degree vertices as shown in the figure then we have a cycle. Euler showed that for a graph G, where all nodes have even degree, it is possible to have a tour that uses each edge once. This means the every even degree graph has a tour that uses each edge once. We want to translate that MST into a graph with each vertex having even degree. The way to do this is as follows.

```
Find the MST of G
Find the Minimum Cost Matching (MCM) on odd degree vertices of MST
The Tour, T is a Eulerian walk of the MST and MCM
```

The idea is that the cost of the tour is $C(T) = C(MST) + C(MCM)$. The optimal TSP costs less than the MST. And the cost of the MCM is at most half the cost of the best tour, since we can get a matching from any tour by taking every other segment. Thus, this algorithm yields a 3/2-approximation.

Papadimitriou and Vempala showed that approximating TSP to better than a factor of 1.01 is NP-complete.

Theorem 19.3 *(Christofides) There is a 3/2-approx algorithm for MIN-TSP [2].*

Proof:

Let T be an optimal tour and let M be an MST of G. Consider the set of vertices of odd degree in M. There must be an even number of such vertices (the sum of the degrees of the vertices in any undirected graph must be even). Now, find a minimum weight perfect matching U of the odd degree vertices and let V be the union of M (the MST) and U (the perfect matching). (Note that V is now a multigraph). Then by construction, every vertex in V has even degree.

Now, any such graph has an Eulerian tour which covers each edge exactly once. Since this is a multigraph, it may be the case that the same edge is counted more than once. Short-circuit these edges (as before) to get a tour T^*. It is clear that $|T^*| \leq |M| + |U|$. Recall that $|M| \geq |T|$. Also note that we must have $2 \times |U| \leq |T|$ because taking every other edge of a tour defines a matching. Take the complement of that matching (with respect to the tour) to get another matching. Now notice that each of these matchings must be at least as long as the best matching. It follows that

$$|T^*| \leq |M| + |U| \leq |T| + 1/2|T| = 3/2|T|.$$

Hence the algorithm is indeed a 3/2-approximation. It is not hard to find an example where the approximation ratio is exactly 3/2, so the analysis is tight.

19.4 Maximum Clique

A clique in a graph G is a set of vertices, every pair of which is connected by an edge. In other words, a clique is a set of vertices for which the induced graph is a complete graph. The problem of maximum clique is to find the largest clique in a given graph. Unfortunately, this problem does not lend itself to approximation. To attain an approximation ratio better than $O(n^{.99})$ is NP-complete. In other words, if the maximum clique is K^*, we will not be able to find a clique K such that $n^{.99}|K| >= |K^*|$. This is a very discouraging result.

Nevertheless, there is a nontrivial approximation algorithm for the maximum clique problem. Divide the n vertices into n/k groups, each of size k. For each group, we check all subsets to see if they are cliques. Finally, we just return the largest clique that we found. Estimating conservatively, this procedure takes $O(nk2^k)$ time. Thus, picking $k = O(\log n)$ yields a procedure that takes polynomial time.

The success of this algorithm relies on the fact that any subset of a clique is a clique itself. Let K^* be the maximal clique in the graph. By the pigeonhole principle, at least one of the n/k groups must contain at least $(k/n)|K^*|$ of the vertices of K^*. Thus, letting K be these vertices, our approximation algorithm finds a clique satisfying $|K| \geq (k/n)|K^*|$. This is an approximation ratio of $O(n/\log n)$.

19.5 Set Cover

Let $[n]$ denote {1, 2, ..., n}, and let $\mathcal{T}$ be a set of subsets $S_1, S_2, \ldots, S_m \subset [n]$. The problem of set cover is to find C^*, the smallest subset of $\mathcal{T}$ that covers each of the elements in $[n]$.

The greedy algorithm is simple. Each step, pick the set that contains the largest number of elements that have not yet been covered. We can construct an example where the greedy algorithm does not approximate the optimal within a constant ratio.

$$\begin{aligned} S_1 &= \{1, 3, 5, \ldots, n-1\} \\ S_2 &= \{2, 4, 6, \ldots, n\} \\ S_3 &= \{1, 2, 3, \ldots, (2/3)n\} \\ S_4 &= \{(2/3)n, (2/3)n+1, \ldots, (2/3)^2 n\} \\ S_5 &= \{(2/3)^2 n, (2/3)n+1, \ldots, (2/3)^3 n\} \\ &\ldots \end{aligned}$$

The greedy algorithm picks S_3 in the first round, then S_4, then S_5, and so forth. In the end, $O(\log n)$ sets have been selected. However, the optimal score here is 2, since S_1 and S_2 together cover $[n]$. Thus, the best approximation ratio of the greedy algorithm in this example is $O(\log n)$. In fact, the greedy algorithm never performs any worse than a multiple of $O(\log n)$ times the optimal performance.

Theorem 19.4 *For the greedy set cover C, $|C| \leq |C^*| H(\max_i |S_i|)$.*

Here H is the harmonic function.

$$H(m) = \sum_{j=1}^{m} \frac{1}{j} = O(\log m)$$

The proof is somewhat complicated.

Let C^* be the optimal solution to the set cover problem, C be the greedy solution, $c = |C|$, and $C = \{S_{i_1}, S_{i_2}, S_{i_3}, \ldots, S_{i_c}\}$. Let $P(x)$ be the number k such that $x \in S_{i_k}$ but $x \notin S_{i_j}$ when $j < k$. Let $P^{-1}(i)$ denote the set $\{y : P(y) = i\}$. Thus, the c sets $P^{-1}(i)$ are a proper partition of $[n]$.

Now let S_i be one of the sets in C^*. Define

$$\begin{aligned} T_{m,i} &= \{x \in S_i : |P^{-1}(P(x))| \leq m\} \\ p &= \min_{x \in T_{m,i}} P(x) \end{aligned}$$

Claim 19.5 *$|T_{m,i}| \leq m$.*

Proof:

At step p of the greedy algorithm, S_{i_p} was chosen when S_i was available. Thus, S_i must have no more elements than S_{i_p} does that are not covered by one of $S_{i_1}, S_{i_2}, \ldots, S_{i_{p-1}}$. Thus, $|T_{m,i}| \leq |P^{-1}(p)|$. Since $\exists x \in T_{m,i}$ such that $P(x) = p$, we know $|P^{-1}(p)| \leq m$. Transitively, this means $|T_{m,i}| \leq m$.

Now, define

$$\begin{aligned} f(x) &= |P^{-1}(P(x))|^{-1} \\ F(S) &= \sum_{x \in S} f(x) \end{aligned}$$

As a result of the claim above, there is at most one element $x \in S_j$ such that $f(x) \geq 1$. Similarly, there are at most two $x \in S_j$ such that $f(x) \geq 1/2$, at most three $x \in S_j$ such that $f(x) \geq 1/3$, et cetera. Thus, $F(S_j) \leq H(|S_j|) \leq H(\max_i |S_i|)$.

Since each element in $[n]$ is contained in one of the sets of C^*,

$$F([n]) \leq \sum_{S \in C^*} F(S) \leq |C^*| H(\max_i |S_i|)$$

$[n]$ can also be partitioned into the $P^{-1}(i)$. By the definition of f, $F(P^{-1}(i) = 1$ for all i. Thus,

$$F([n]) = \sum_{i=1}^{c} F(P^{-1}(i)) = |C|$$

Transitively, we have $|C| \leq |C^*| H(\max_i |S_i|)$ and the theorem is proven.