Approximation Algorithms

Set Cover

Recall the Set Cover problem: Given n items and a family of m sets, $S_1, S_2, \ldots, S_m$, find the minimum size collection of these sets $C \subseteq S_1, \ldots, S_m$ that "covers" all n items. In other words, we want to find C such that $\forall n, n \in \bigcup_{S_i \in C} S_i$ and $|C|$ has the minimum size.

It is not known how to solve this problem optimally in polynomial time. We will settle for an approximate solution to this problem, and in particular we will see an algorithm to find a solution to this problem that uses no more $\ln n$ times the optimal number of sets in the family.

There are a couple algorithms that achieve this bound. One simple algorithm uses a Greedy approach.

Algorithm: On each iteration, pick the set that covers the most uncovered vertices.

The tricky part of this algorithm is its analysis.

Analysis: Each iteration of this algorithm adds an additional set to the cover. To prove that this algorithm achieves a $\ln n$-approximation, we may therefore analyze the number of iterations this algorithm performs. First we will establish an equation for the number of uncovered elements after a given number of iterations. Suppose the optimum cover uses k sets. At each iteration there must exist some set that covers a $1/k$ fraction of the remaining uncovered elements. Therefore, if the number of uncovered elements at the beginning of the iteration was r, the number of uncovered elements at the end of the iteration is at most $r(1 - 1/k)$. We initially start this algorithm with n elements uncovered, and after j iterations the number of remaining elements is $n(1 - 1/k)^j$.

We can use this expression to determine the total number of iterations performed. First, note that $(1 - 1/k)^k \leq 1/e$ for all positive k. With this inequality, we find that $(1 - 1/k)^j = ((1 - 1/k)^k)^{j/k} \leq (1/e)^{j/k}$. Therefore, the number of uncovered elements after j iterations is at most $n(1/e)^{j/k}$. Second, we realize that if the number of uncovered elements ever falls below 1, then all elements are covered and this algorithm will terminate. Therefore we are interested in the case where $ne^{-j/k} < 1$. This condition occurs when $j = O(k \ln n)$, and therefore this algorithm will terminate in at most $O(k \ln n)$ iterations. Each iteration adds one set to the cover, so the cover this algorithm finds contains $O(k \ln n)$ sets. Because k is the number of sets OPT uses to cover all elements, this algorithm provides a $\ln n$-approximation.

Another way to solve this problem involves formulating it as an Integer Linear Program, and relaxing that ILP to a regular linear programming problem. How might this work? (Don't worry, you don't need to know this for 6.046.)

Ideally, we would like to be able to do better than $O(\log n)$ times the optimal solution for set-cover. Unfortunately it was recently proven that it is impossible to get a better approximation on set-cover than $O(\log n)$. Therefore, this algorithm achieves an asymptotically optimal solution.

TSP

Recall the 2-approximation algorithm for the metric traveling salesman problem, which was described in lecture. This algorithm computed a minimum spanning tree of the given graph, picked a root in that tree, and traversed that tree, inserting shortcuts as appropriate to avoid traversing a vertex multiple times. We can modify this algorithm slightly to improve this approximation bound.

Again, consider the MST of the graph. We will again create a traversal of the graph using a traversal of this MST with some shortcuts. What we will change from our 2-approximation algorithm is how these shortcuts are selected. The idea is to select these shortcuts such that their total weight is at most $w(OPT)/2$. Because the weight of the edges in the MST is at most $w(OPT)$, adding these shortcuts to the MST will give us a selection of edges of weight at most $\frac{3}{2}w(OPT)$. We will then find an Eulerian tour on this graph, which uses each selected edge exactly once, and correct any repeated vertices using the same shortcutting trick as before.

Therefore we must select a set of shortcuts such that, when combined with the MST of the graph, an Eulerian tour exists among all selected edges, and the total weight of these shortcuts must not exceed half of the weight of the optimal tour. How do we do this? Using something known as Christofides' Heuristic.

We will use a minimum-cost matching to find a good set of shortcuts. To enable the existence of an Eulerian tour in a spanning tree, we must add or duplicate edges to ensure that every vertex in the spanning tree has an even degree. We do this by adding an odd number of edges to each odd-degree vertex in the spanning tree, and we find the edges to add by finding a minimum-cost matching of the odd-degree vertices.

To see why this technique works, consider the optimal solution to metric TSP. The optimal solution is a cycle through all vertices in the graph. We can create from the optimal solution a cycle through only the vertices in the graph with an odd degree in the spanning tree, and by the triangle inequality this cycle-through-odds will have cost at most equal to $w(OPT)$. We know that there are an even number of odd-degree vertices in the spanning tree, and by selecting every other edge in the cycle-through-odds we may arrive at two disjoint matchings of the odd-degree vertices. The union of these two matchings is at most $w(OPT)$, and therefore one of these two matchings must cost at most $\frac{1}{2}w(OPT)$. Consequently, the minimum-cost matching between the odd-degree vertices must cost no more than $\frac{1}{2}w(OPT)$, and therefore our selection of shortcuts combined with the edges in the minimum spanning tree must cost no more than $\frac{3}{2}w(OPT)$. Therefore this algorithm gives a $\frac{3}{2}$-approximation for metric TSP.