18.433 Combinatorial Optimization

Approximation Algorithms

November 20,25 *Lecturer: Santosh Vempala*

1 Approximation Algorithms

Any known algorithm that finds the solution to an NP-hard optimization problem has exponential running time. However, sometimes polynomial time algorithms exist which find a "good" solution instead of an optimum solution.

Given a minimization problem and an approximation algorithm, we can evaluate the algorithm as follows. First, we find a lower bound on the optimum solution. Then we compare the algorithm's performance against the lower bound. For a maximization problem, we would find an upper bound and compare the solutions found by our approximation algorithm with that.

1.1 Minimum Vertex Cover

Remember a vertex cover is a set of vertices that touch all the edges in the graph. The Minimum Vertex Cover Problem is to find the least-cardinality vertex cover.

A lower bound on the minimum vertex cover is given by a maximal matching. Since no two edges in a matching share the same vertex, there must be at least one vertex in the vertex cover for each edge in the matching.

Also, notice that the set of all matched vertices in a maximum matching is a vertex cover. This follows as any edge whose end-vertices are both unmatched may be added to the matching, contradicting the maximality of the matching. Clearly this algorithm contains twice as many vertices as our lower bound, which is the number of edges in a maximal matching. So the algorithm is within twice optimal.

Two issues are of interest here: how good is our lower bound with respect to the optimal solution, and how good is our final solution with respect to the optimal solution.

First we show that the lower bound can be a factor 2 away from optimal. Consider the complete graph with n edges. The maximal matching has $\frac{n}{2}$ edges, so our lower bound is $\frac{n}{2}$.

However, $n-1$ vertices are required to cover the graph. To see this, consider any set of $n-2$ vertices. Because the graph is complete, there is an edge between the two omitted vertices that is not touched by the $n-2$ chosen vertices. For large n, we have $\frac{\text{OPT}}{\text{LB}} = \frac{n-1}{\frac{1}{2}n} \to 2$. So by comparing any algorithm to this bound, we will never have a tighter result than within twice optimal.

Now we compare our final solution to the optimal one. Our algorithm outputs all the vertices matched by a maximal matching. So consider a complete bipartite graph, with n vertices on each side of the partition. The graph contains a perfect matching so the algorithm outputs every vertex i.e. $2n$ vertices. The optimal vertex cover needs only n vertices, though, those vertices from one side of the partition. Thus we see that the bound on the algorithm's performance is tight.

1.2 The Travelling Salesman Problem

The travelling salesman problem is the following. Given a complete graph $G = (V, E)$, and a metric function $d(i, j)$ that gives edge lengths, find a Hamiltonian cycle of minimum total length. Notice that a minimum spanning tree (MST) is a lower bound on the optimum. For suppose there was a tour shorter than the MST. Remove an edge from the tour, and then it is a smaller spanning tree.

The Tree Algorithm: We can also construct an approximate solution from the MST. Find the MST, and double the edges. Note that each vertex now has even degree. Thus we can find an Eulerian tour in our new graph. From this tour we can derive an Hamiltionian cycle by short-cutting past vertices that we have already visited. By the triangle inequality, which holds because $d(i, j)$ is a metric, the cost of the Hamiltonian cycle is at most the cost of the Eulerian tour. The cost of the Eulerian tour, though, is twice the cost of the MST. Therefore our algorithm provides a solution that is within twice optimal.

Thus our algorithm gives at worst a factor 2 approximation. Is this the true performance of the algorithm? Again, let us first compare the value of the lower bound to the optimal solution. There are cases where the factor 2 may be obtained. Consider the graph consisting of a path on n vertices. Let the edge costs on the path be 1. The cost of edges not on the path are given by their shortest path distances along the path. Thus the minimum spanning tree is of length $n-1$. The minimum tour, though, has length $2(n-1)$. Thus even an algorithm that finds the true solution seems only half optimal compared to this bound.

Now let us observe how the algorithm actually performs against the optimal solution. Again,

however, the algorithm may provide a solution that is twice optimal. Consider a ladder-shaped graph, with edge weights of 1. All other edges also assigned weights according to the shortest path distances. Note that the diagonal edges from one rung to the next have weight weight 2. One minimum spanning tree is all the rungs and one side of the ladder. These edges form a comb-shape. Then one way the algorithm could run is to follow the comb-shaped MST, but short-cut between rungs in a saw-tooth fashion. At the end of the ladder, jump back to the start. If there are n rungs, the length of this tour is $4n - 2$. However, the shortest tour is to run around the perimeter of the ladder, avoiding all rungs. This has length $2n$.

Christofides' Algorithm : Another feasible bound on the optimum is given by a minimum-weight perfect matching. Since this is the collection of shortest distances between two points, and contains $\frac{1}{2}n$ edges since our graph is complete, the minimum-weight perfect matching is less than half the optimum tour. In fact the minimum-weight perfect matching on any even subset of edges is also less than half the optimum tour, by the triangle inequality. This suggests a new algorithm.

First, find an MST. Then find a minimum-weight perfect matching on the odd-degree vertices. Since we added one edge per odd-degree vertex, all vertices are now even degree and we can find an Eulerian tour. Short-circuit the tour to produce an Hamiltonian cycle. The length of this cycle is less than the sum of the length of the MST plus the length of the min weight perfect matching. This is less than three-halves times the optimum. Thus we obtain a $\frac{3}{2}$-approximation algorithm. This simple algorithm provides the best guarantee known for the metric Travelling Salesman Problem.

1.3 Set Cover

Consider a set S of elements $\{e_1, e_2, e_3, ..., e_n\}$, and subsets $S_1, S_2, S_3, ..., S_m \subseteq S$. The Set Cover Problem is to find a minimum collection of subsets whose union equals S.

This problem can be written in matrix form. Let the rows represent subsets S_i and the columns represent elements e_j, and let $M_{i,j}$ equal 1 if $e_j \in S_i$ and 0 otherwise. Then the problem is to find the smallest-cardinality set of rows that covers all the columns.

One approximation algorithm is the greedy algorithm. At each step, pick the subset S_i that covers the most uncovered elements. We give an example to show that this algorithm can not give a performance guarantee better than $O(\log n)$. Let $S = \{e_1, e_2, ...e_{2n}\}$, with $S_1 = \{e_1, e_2, ...e_n\}$ and $S_2 = \{e_{n+1}, e_{n+2}, ...e_{2n}\}$. Then, of course, S_1 and S_2 are a set cover.

Now let S_3 contain the first half of both S_1 and S_2 and one more element, so that it covers more than S_1 and S_2. Let S_4 contain the next quarter of both S_1 and S_2 so that, after picking S_3, it covers slightly more of the rest of the S than S_1 and S_2. Continue defining subsets S_i in this fashion. The greedy algorithm will pick $S_3, S_4, \ldots$, and end up with up to $\log_2(\frac{n}{2})$ sets, whereas the optimum is only two sets.

In order to facilitate analysis of this algorithm, assign a cost to each element based on the set that the greedy algorithm picks. Let S_k be the k^{th} set chosen by the greedy algorithm, and let S'_k be the set of elements in S_k that were not previously covered by the sets $\{S_1, S_2, \ldots, S_{k-1}\}$. Now let the cost of an element e_i to be $\frac{1}{|S'_k|}$, where S_k is the first set to cover e_i. It follows that the sum of the costs of the elements is the cardinality of the set cover.

Note that the best-possible average cost of an element is $\frac{\text{OPT}}{n}$. In addition, since the greedy algorithm takes the least-cost elements first, we know that $\text{cost}(e_1) \leq \frac{\text{OPT}}{n}$. Now, OPT is also an upper bound on the cost of the remaining elements, so $\text{cost}(e_k) \leq \frac{\text{OPT}}{n-k+1}$. So the greedy cost is

$$\sum_{k=1}^{n} \text{cost}(e_k) \leq \sum_{k=1}^{n} \frac{\text{OPT}}{n-k+1} = \text{OPT} \sum_{k=1}^{n} \frac{1}{n-k+1}$$

We can bound the summation on the right by integrals, and thus observe that the sum is between $\ln(n+1)$ and $\ln n + 1$. So the greedy algorithm is within $\ln n + 1$ of the optimum. This bound is tight by the previous example.

It has been shown that approximating Set Cover to better than $O(\log n)$ is itself NP-hard.

2 Relax and Round

A general approximation technique is the following. First model the problem as an integer program. Then relax the constraints to obtain a linear program. Solve the linear program (perhaps by using a separation oracle). Then round the fractional solution to an integral solution.

2.1 Minimum Congestion

Given a graph $G = (V, E)$, and a set of pairs of vertices $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$, find a path between each pair s_i and t_i. The *congestion* on an edge is the number of paths that use the edge. The problem is to find a set of paths that minimizes the maximum congestion.

This problem is NP-complete.

To reduce this problem to an integer program, use variables $x^i_{j,k} \in \{0, 1\}$. Assign 1 when the edge (j, k) is included in the path from s_i to t_i, and 0 otherwise.

In order to ensure that we generate paths, we set up a flow problem. We require a flow of value 1 from each s_i to t_i. Then the divergence of a vertex is 0, i.e. $\sum_k x^i_{k,j} = \sum_l x^i_{j,l}$ for each vertex j, except when $j = s_i$ or $j = t_i$, in which case the divergence is 1.

Treat the objective function $\max_{e \in E} \sum_i x^i_e$ as another constraint, such that all the congestions are less than some integer c. Then we can find the optimum by using this feasibility problem in a binary search. The constraint is then $\sum_i x^i_e \leq c$ for all edges e.

There is an integrality gap, that is, the optimal value of the integer program is not the same as the optimal value of its relaxation. Consider the graph of a box, with s_1 in the upper left corner, s_2 in the upper right, t_1 in the lower right and t_2 in the lower left. Then the optimum solution of the integer program has congestion 2, whereas we can assign flow of $1/2$ to all the edges for each i and obtain maximum congestion of 1.

Our first approach to convert the linear program result to a set of paths might be to set x^i_e to 1 with probability x^i_e and to 0 with probability $1 - x^i_e$. Then the expected value is $E(x^i_e) = 1 \cdot x^i_e + 0 \cdot (1 - x^i_e) = x^i_e$, and the expected congestion is the sum of the expected edge weights, which is $\sum_i x^i_e$. However, this may not be a solution to the problem: this algorithm could lead us to take a set of edges that do not form paths from s_i to t_i.

Notice that the solution to the linear program gives a set of flows, rather than paths. We can decompose the flows into a sum of paths by the following: find a path from s_i to t_i, and set its weight to the minimum capacity λ^i_1 of its edges. Then delete $\lambda^i_1 p_1$ from the flow and repeat, to get λ^i_j's.

Also, notice that $\sum_j \lambda^i_j = 1$ for all i. Therefore, our process for converting the linear program result to an integer solution will be to pick a path for each i with probability λ^i_j. Then, since the sum of the λ^i_j for a given edge is x^i_e, the expected value of paths from s_i to t_i using edge e is x^i_e. Therefore the expected congestion per edge is still $\sum x^i_e$. Notice that this is the expected congestion for a given edge, and we do not yet know what the maximum congestion will be.

Suppose that for any particular edge, given its integer (relaxed) congestion X, and its expected congestion μ, we could show that the probability of X being greater than some constant factor $c\mu$ is less than $\frac{1}{n^2}$, where n is the number of vertices. Then (union bound), with probability at least $1/2$, *every edge* has a congestion less than $c\mu$. Since the expected

congestion of the linear program is less than the congestion of the integer program, $c\mu \leq c\,\mathrm{OPT}$.

Markov's inequality says that if X is a non-negative random variable, then $P(X > c\mu) < 1/c$ for $c > 0$, so we could use $c = n^2$. However, this is no better than what we could have done picking random paths. So, for a particular edge, let X^j be the indicator of the event "path j through the edge is chosen in the rounding", let $X = \sum_j X^j$ be the integer congestion of the edge, and let $\mu = E(X)$. Then

$$P\left(X > (1+\delta)\mu\right) = P\left(e^{tX} > e^{t(1+\delta)\mu} \cdot \frac{E(e^{tX})}{E(e^{tX})}\right)$$

We know

$$E(e^{tX}) = E(e^{t\sum X^i}) = \prod_i E(e^{tX^i}) = \prod_i (p_i e^t + 1 - p_i) = \prod_i (1 + p_i(e^t - 1))$$

Since $1 + x \leq e^x$ for any real x, and $\sum p_i = \mu$, we have

$$E(e^{tX}) \leq \prod_i e^{p_i(e^t-1)} = e^{e^t-1} \prod e^{p_i} = e^{\mu(e^t-1)}$$

¿From Markov's inequality, we obtain

$$P\left(e^{tX} > e^{t(1+\delta)\mu} \cdot \frac{E(e^{tX})}{E(e^{tX})}\right) \leq \frac{E(e^{tX})}{e^{t(1+\delta)\mu}}$$

By substitution

$$\frac{E(e^{tX})}{e^{t(1+\delta)\mu}} \leq \frac{e^{\mu(e^t-1)}}{e^{t(1+\delta)\mu}}$$

Set $t = \ln(1+\delta)$. For $1 + \delta \geq 2e$, we get

$$P\left(X > (1+\delta)\mu\right) \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \leq \left(\frac{e^{1+\delta}}{(2e)^{1+\delta}}\right)^\mu = \frac{1}{2^{(1+\delta)\mu}}$$

Suppose we take δ such that $(1+\delta)\mu = \max(2e\mu, 2\log n)$. Then

$$P(X < \max(2e\mu, 2\log n)) \leq \frac{1}{2^{2\log n}} = \frac{1}{n^2}$$

Therefore with probability $\frac{1}{2}$ none of the the edges have congestion greater than $\max(2e\mu, 2\log n)$. It follows that, by repeating the rounding procedure, we obtain an $2e\mathrm{OPT}+2\log n$-approximation guarantee factor. Thus in the worst case we have an $O(\log n)$ guarantee.