COM S 783 / OR&IE 634 Approximation Algorithms **Fall Semester, 2003**

Lecture 1: Sept 1, 2003

Lecturer: Yuval Rabani

1 Introduction

This course deals with problems in *combinatorial optimization*. What we mean by this term is that we are given, implicitly, a finite set of weighted objects (for example, the set of all spanning trees of a weighted graph), and we want to find an object of minimum or maximum weight (for example, a minimum cost spanning tree). Such an object is an *optimum* of the function mapping objects to their weights, and the process of finding an optimum is called *optimization*. Thus, an instance of a combinatorial optimization problem is defined by a finite structure that induces a finite set of objects (or *feasible solutions*), an implicit function that maps feasible solutions to their weights, and an optimization criterion (minimum or maximum). Let F denote the finite structure, let $\mathcal{S}$ denote the induced set of feasible solutions, and let w denote the weight function. In dealing with computational problems, we need to place restrictions on the representation of these items. Firstly, we require that the size of each $S \in \mathcal{S}$ is polynomially bounded in the size of F. Next, we require that there be a polynomial time algorithm that on input S, F decides if indeed $S \in \mathcal{S}$. Finally, we require that there be a polynomial time algorithm that on input S, F, where $S \in \mathcal{S}$, computes $w(S)$. Such problems are called NP optimization problems, because their corresponding decision problems (is there an object with weight below/above a given threshold?) are in NP.

For some combinatorial optimization problems there are polynomial time algorithms that find an optimum. This is more the exception than the rule. Most interesting problems are NP-hard (usually, their decision versions are NP-complete). In particular, this usually implies the absence of good characterizations for the optimum. In this course, we study one approach to tackling such problems, both structurally and computationally. This approach calls for the design and analysis of *approximation algorithms*, i.e., polynomial time algorithms that find a solution whose weight is close to optimum. In fact, finding such approximately optimal solutions is often NP-hard too. Thus, much of the work in this field is devoted to the exploration and discovery of the boundaries of polynomial time computation. We want to find out the best approximation guarantees that are possible under the assumption that P$\neq$NP.

The secret to designing approximation algorithms is the discovery of a polynomially time computable lower bound (in the case of minimization problems) or upper bound (in the case of maximization problems) on the optimum. We hope to avoid the complexity of characterizing the optimum itself by bounding it imprecisely. The tightness of the bound will determine the quality of the approximation derived from this bound. In fact, we know that unless P=NP, the gap between the bound that we get and the optimum must vary from instance to instance, otherwise we could compute the value of the optimum. Thus, when our bound is far from the optimum, our approximation will actually yield a solution close to optimum, and when our bound is close to optimum, our approximation will deviate from it more significantly.

2 Examples

Consider an NP-optimization problem and a proposed (approximation) algorithm $\mathcal{A}$ for that problem. Let $S^* = S^*(F)$ denote an optimal solution for input F, and let $S = S(F)$ denote the output of $\mathcal{A}$ for the same input. For minimization problems, we say that $\mathcal{A}$ is a c-approximation algorithm iff this for all F, $w(S)/w(S^*) \leq c$. For maximization problems, the condition is $w(S)/w(S^*) \geq c$.

The traveling salesperson problem (TSP). In this problem we are given a weighted complete graph on n nodes. Our goal is to find a tour, i.e. a simple cycle that spans all the nodes, of minimum total edge weight. The following theorem is easy

Theorem 1 (Sahni and Gonzalez [3]). For any bound $B > 0$, the existence of a polynomial time B-approximation algorithm for TSP implies that P=NP.

Proof: The proof is by reduction from Hamiltonian Tour. Consider an instance $G = (V, E)$ to Hamiltonian Tour. Construct a TSP instance as follows: Put $n = |V|$. For every ij, put $w(ij) = 1$ if $ij \in E$ and $w(ij) = nB$ otherwise. Now, G is Hamiltonian iff there is TSP tour of weight n, and G is not Hamiltonian iff every TSP tour has weight more than nB. (Both conditions are necessary and sufficient.) Thus, a B-approximation algorithm for TSP can decide hamiltonicity. $\square$

Metric TSP. A useful restriction on the TSP problem is to require that the weights form a metric. Formally, we require the triangle inequality: For all i, j, k, $w(ij) + w(jk) \geq w(ik)$. (Notice that the weight function is defined on unordered pairs.) This restricted problem is called Metric TSP. Despite the restriction, it is still NP-hard. A simple lower bound on the cost of an optimal solution is the cost of a minimum spanning tree (MST). This is because an optimal solution is a simple cycle, so it contains a path. This leads to the following approximation algorithm: Compute a minimum spanning tree. Double each edge. The resulting graph is Eulerian, as all the degrees are even. Take an Euler tour and use the triangle inequality to shortcut the tour whenever a node is revisited. Let OT denote the cost of the tour we find, and let ET denote the cost of the Euler tour in the doubled tree. We have

$$\mathrm{OT} \leq \mathrm{ET} \leq 2 \cdot \mathrm{MST} \leq 2 \cdot \mathrm{TSP}.$$

Thus, we have designed a 2-approximation algorithm for metric TSP. This result is due to Rosenkrantz, Stearns, and Lewis [2]. Christofides [1] proposed the following improvement:

Theorem 2 (Christofides [1]). There is a polynomial time $\frac{3}{2}$-approximation algorithm for metric TSP.

Proof: Instead of doubling the edges of the MST, add a minimum cost matching MM for the odd-degree nodes of the tree. Adding MM also creates an Eulerian graph, and we can take the Euler tour and shortcut it as before. To analyze this algorithm we will need another lower bound. Firstly, notice that an optimal TSP tour on a subset of the input nodes cannot be longer than the optimum on the entire node set. Secondly, notice that every tour on an even number of nodes decomposes into two perfect matchings, and therefore the minimum cost matching on the same node set has weight at most half the weight of the TSP tour. Combining these observations, using subTSP to denote the cost of an optimal TSP tour on the subset of nodes that participate in MM, and using CT to denote the cost of the tour output by the algorithm, we get

$$\mathrm{CT} \le \mathrm{MST} + \mathrm{MM} \le \mathrm{TSP} + \frac{1}{2}\mathrm{subTSP} \le \frac{3}{2}\mathrm{TSP}. \qquad \square$$

References

[1] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report no. 388, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

[2] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 1977.

[3] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. Assoc. Comput. Mach.*, 23:555–565, 1976.