6.893 Approximability of Optimization Problems	September 13, 1999
Lecture 2	
Lecturer: Madhu Sudan	*Scribe: Alantha Newman*

Today we will discuss and give examples of NP-hard optimization problems that can be "reasonably" approximated in polynomial time by non-trivial algorithms.

1 TSP

The first example of a problem with a non-trivial approximation algorithm is the Traveling Salesperson Problem. The best known approximation for this problem is $\frac{3}{2}$. Below we give a 2-approximation algorithm for this problem. The Traveling Salesperson Problem is defined as follows:

Input: $G = (V, E, d)$, where $d : E \rightarrow Z^+$ is a distance function on the edges of G

Goal: Find a tour, i.e. sequence of vertices $v_1, \ldots v_m$, such that: every vertex appears at least once in the sequence; (v_i, v_{i+1}) is actually an edge of the graph; and $v_1 = v_m$.

Objective: Minimize $\sum_{i=1}^{m-1} d(v_i, v_{i+1})$

A lower bound for the length of an optimal TSP tour of G is the cost of a Minimum Spanning Tree (MST) of G. In other words, $MST \leq OPT_{TSP}$. The proof of this is simple: If you remove any edge from a legitimate TSP tour, you obtain a spanning tree.

We can find an MST in $O(m \log m)$ time using Kruskal's algorithm. Given an MST, we can take an Eulerian tour of the MST, which will give us a tour of the graph which is $\leq 2MST \leq 2OPT_{TSP}$. If we assume the triangle inequality, i.e. $\forall i, j, k \in V, d(i,j) + d(j,k) \leq d(i,k)$, we can visit all the edges exactly once by taking shortcuts from each vertex in the tour to the next unvisited vertex in the tour.

This approximation algorithm exhibits the necessary ingredients of all approximation algorithms:
1) Finding an efficiently computable lower bound on the cost of the optimal solution.
2) Find a solution whose value is $\leq \alpha$ (Lower Bound).

2 Vertex Cover

This methodology is exhibited again in the construction of an approximation algorithm for the Minimum Vertex Cover problem. Here we will consider an unweighted graph.

Input: $G = (V, E)$

Goal: Find $C \subseteq V$ such that $\forall (u, v) \in E$, either u or v is in C.

Objective: Minimize $|C|$

First we note that if M is a maximal matching for G, then $|M| \leq OPT_{VC}$. Take any maximal matching; for each edge in the matching at least one endpoint must be in the Vertex Cover. If G has a maximal matching of size k, then G has a Vertex Cover of size $2k$; for each edge in the matching, pick both endpoints to be in the Vertex Cover. This gives us a Vertex Cover that is $2|M| \leq 2OPT_{VC}$. We can find a maximal matching using a greedy algorithm. It is an open problem as to whether or not 2 is the best approximation ratio that can be obtained for Vertex Cover.

3 Knapsack

Another problem for which there exists "reasonable" approximation algorithms is Knapsack. The Knapsack problem is defined below:

Input: n objects. Object i has weight w_i and value v_i. $w_1 \le \cdots \le w_n \le C$, where C is the capacity of the knapsack.

Goal: Find a subset of the n elements that does not exceed capacity C, i.e. find $S \subseteq \{1, \dots n\}$ $\sum w_i \le C$.

Objective: maximize $\sum_{i \in S} v_i$

We will first give a pseudo-polynomial algorithm for Knapsack. Assume $v_1, \dots, v_n$ are non-negative integers and $\sum v_i = V_{TOTAL}$. We will give an algorithm with running time poly(n, V_{TOTAL}). This is not necessarily polynomial since V_{TOTAL} can be exponential in n.

The dynamic programming formulation for this problem follows. Define $f(k, U)$ to be the minimum weight collection of objects from $\{1, \dots k\}$ that have value $\ge U$. Then

$$f(k, U) = \min \{f(k-1, U), f(k, U - v_k) + w_k\}$$

Even if V_{TOTAL} is exponential in n, we can use this algorithm to obtain a solution of value $(1 - \epsilon)OPT_{VALUE}$ that runs in time poly$(\frac{1}{\epsilon}, n)$. We will create a new instance of the problem that has only $\frac{1}{\epsilon}$ different values for the items, so that we can solve this instance in time poly$(\frac{1}{\epsilon}, n)$. An optimal solution for this new instance will be a $(1-\epsilon)$-approximation to the optimal solution for the original problem.

The algorithm is as follows: divide the objects according to their value into $\frac{1}{\epsilon}$ sections. We assume $\frac{1}{\epsilon}$ is an integer. Each section has width ϵV_{TOTAL}, i.e. it includes all the objects with $k\epsilon V_{TOTAL} \le v_i \le (k+1)\epsilon V_{TOTAL}, {}_{0 \le k \le \frac{1}{\epsilon}-1}$. For each object in section k, assign value $k\epsilon V_{TOTAL}$.

Let S denote the instance of the problem with $\frac{1}{\epsilon}$ different values. Let S' denote the original problem instance. Then: $OPT_S + \epsilon n V_{TOTAL} \ge OPT_{S'}$. We also know that $V_{TOTAL} \le n(V_{MAX}) \le n(OPT_S)$. So $OPT_S(1 + \epsilon n^2) \ge OPT_{S'}$. If we choose $\epsilon = \frac{\delta}{n^2}$, then we will obtain a solution with value $(1-\delta)OPT_{S'}$.

This algorithm is an example of an FPTAS.

4 FPTAS

A problem π has a Fully Polynomial Time Approximation Scheme if $\exists$ alg A that takes as input an instance x of π and a parameter δ runs in time poly$(|x|, \frac{1}{\delta})$ and produces $(1+\delta)$-approximate solution.

5 Strong NP-completeness

Consider optimization problems that have non-negative integer objectives. Let $\pi_p = \{$ instances of a problem π whose optima lie between $0 \dots p(n)\}$. This is the set of polynomially bounded optimization problems. A problem π is *Strongly NP-complete* if $\exists$ polynomial p such that π_p is NP-complete. Knapsack is not Strongly NP-complete.

Proposition 1 *If π is Strongly NP-complete then π does not have FPTAS.*

Proof For $\forall p$, let $\delta = \frac{1}{2p(n)}$. Then you can get an optimal solution in time poly$(\frac{1}{\delta}, n) =$ poly(n), which is a contradiction to the fact that π is Strongly NP-complete. ■

6 Minimize Makespan

Input: n jobs: $t_1, \ldots, t_n$; and m processors

Goal: Partition the n jobs into m sets $S_1, \ldots, S_m$.

Objective: Minimize maximum $_{j \in \{1,\ldots m\}} \sum_{i \in S_j} t_i$

The Minimum Makespan problem has a 2-approximation algorithm due to Ron Graham. This problem is (weakly) NP-complete even for $m = 2$ (reduction from partition). For $m = 2$, there is an FPTAS using an approach similiar to that shown above for Knapsack. In fact, for all constants m, we can obtain an FPTAS in which m is an exponent in the running time. When m is an arbitrary number, the problem becomes Strongly NP-complete because of a reduction from the 3-partition problem. So for arbitrary m, there is no FPTAS. The 3-partition problem is defined as follows:

Input: Elements $t_1, \ldots, t_{3n}; t_i \in \{1, \ldots, n\}; v = \frac{\sum t_i}{n}$

Goal: Find sets $S_1, \ldots, S_n; |S_i| = 3$, such that $\forall i \sum_{j \in S_i} t_j = v$.

Because of the NP-completeness of 3-partition, we can do a $(1+\delta)$-approximation, but not in time poly$(\frac{1}{\delta}, n)$. So there exist algorithms in between constant factor approximation algorithms and FPTAS. For the Minimum Makespan problem, we can find a $(1+\delta)$-approximation in time $n^{O(\frac{1}{\delta})}$.