CMPUT 675: Approximation Algorithms **Winter 2005**

Lecture 1 Introduction : Jan 12, 2005

Lecturer: Mohammad R. Salavatipour *Scribe: Xiaomeng WU*

1.1 Introduction

We will be studying NP-hard optimization problems. For these problems, we would like to:

1. find the optimal solution,
2. find the solution fast, often in polynomial time
3. find solutions for any instance

Unfortunately, with the assumption of $P \neq NP$, we cannot have all the above three at the same time. Therefore, we need to relax at least one of them. If we:

1. relax (1), then we are into study of special cases of the problem.
2. relax (2), we will be in the field of integer programming and the techniques there such as branch-and-bound, etc.
3. relax (3), we are into study of heuristics and approximation algorithms.

We are going to focus on (3) in this course, and in particular, on the field of approximation algorithms. We are interested in:

- finding solutions that are within a guaranteed factor of the optimal solutions, and
- We want to find this solution fast (i.e. polynomial time).

Why is it important to study approximation algorithms?

- We need to solve optimization problems (they appear everywhere!). Saying that because they are NP-hard we don't know of any efficient way of solving them is not enough. If you cannot find the optimal solution, try your best!
- We can prove how good the solution is (this is sometimes critical in applications).
- We can have an understanding of how hard the problem is, and this in turn can help to design better algorithms.
- It is Fun!

1.2 NP Optimization problems

Definition 1.1 *An NP optimization problem, Π, is a minimization (maximization) problem which consists of the following items:*

Valid instances: each valid instance I is recognizable in polynomial time. (note: 'Polynomial time' means polynomial time in terms of the size of the input.) The set of all valid instances is denoted by D_Π. The size of an instance $I \in D_\Pi$, denoted by $|I|$, is the number of bits required to represent I in binary.

Feasible solutions: Each $I \in D_\Pi$ has a set $S_\Pi(I)$ of feasible solutions and for each solution $s \in S_\Pi(I)$, $|s|$ is polynomial (in $|I|$).

Objective function: A polynomial time computable function $f(s, I)$ that assigns a non-negative rational value to each feasible solution s for I.

We often have to find a solution s such that this objective value is minimized (maximized). This solution is called optimal solution for I, denoted by $OPT(I)$.

Examples:

- Vertex Cover (V.C.) problem:

  Given a graph $G(V, E)$ and a function $c : V \to Q^+$ assigning cost to each vertex, find a vertex cover for G with minimum cost. A vertex cover is a subset $V' \subseteq V$ such that every edge of G has at least one end point in V'. A special case when every vertex has unit cost (i.e. c is the unit function) is called the cardinality vertex cover problem. Here, we have:

  valid instances : set of graphs with weighted vertices.

  feasible solutions : all the vertex covers of the given graph.

  objective functions : minimizing the total weight of a vertex cover.

- Minimum Spanning Tree (MST) problem:

  Given a connected graph $G(V, E)$, with each edge $(u, v) \in E$ assigned a weight $w(u, v)$, find an acyclic subset $T \subseteq E$ that connects all the vertices and its total weight is minimized. Since T is acyclic and connects all of the vertices, it is a tree.

  valid instances : a graph with weighted edges.

  feasible solutions : all the spanning trees of the given weighted graph.

  objective functions : minimizing the total weight of a spanning tree.

1.3 Approximation algorithms

An *α-factor approximation algorithm* (or simply an α-approximation) is a polynomial time algorithm whose solution is always within α factor of optimal solution.

Definition 1.2 *For a minimization problem Π, algorithm A has approximation factor α if it runs in polynomial time and for any instance $I \in D_\Pi$ it produces a solution $s \in S_\Pi(I)$ such that $f(s, I) \le \alpha(|I|) \cdot OPT(I)$. α can be a constant or a function of the size of the instance.*

We use $A(I)$ to denote the value of the solution returned by algorithm A for instance I; therefore, from Definition 1.2: $A(I) \leq \alpha(|I|) \cdot OPT(I)$.

Similarly, for a maximization problem, we have

- $OPT(I) \leq \alpha(|I|) \cdot A(I)$, if $\alpha > 1$;
- or $OPT(I) \leq \frac{A(I)}{\alpha(|I|)}$, if $\alpha < 1$.

Definition 1.3 *Algorithm A has asymptotic α-approximation ratio if the ratio* $\lim_{|I|\to\infty} \frac{A(I)}{OPT(I)} \leq \alpha$

Intuitively, for large enough instances, the approximation ratio of the algorithm is almost α (although for small instances it might be larger). Just as there are randomized algorithms that compute exact solutions, there are randomized algorithms that compute approximate solutions.

Definition 1.4 *An algorithm A is a randomized factor α-approximation for problem Π if for any instance $I \in D_\Pi$, A produces a feasible solution s such that $Pr[f(s,I) \leq \alpha(|I|) \cdot OPT(I)] \geq \frac{1}{2}$.*

This probability can be increased to $(1 - \frac{1}{2^x})$ by repeating the same algorithm A for x times.

Definition 1.5 *A (uniform) PTAS (Polynomial Time Approximation Scheme) for an optimization problem is an approximation algorithm A that takes as input not only an instance of the problem, but also a value $\epsilon > 0$, and runs in time polynomial in n and returns a solution s that satisfies: $f(s,I) \leq (1+\epsilon) \cdot OPT(I)$, i.e. it is a $(1+\epsilon)$-approximation algorithm.*

In case of non-uniform PTAS we have a class of algorithms $\{A_\epsilon\}$, one for every possible value of ϵ. A running time $O(n^{\frac{1}{\epsilon}})$ is still polynomial for any fixed ϵ, but computations with ϵ values very close to 1 may turn out to be practically not feasible. This leads to the definition of $FPTAS$, a more restricted version of $PTAS$ and much faster than $PTAS$.

Definition 1.6 *FPTAS(Fully Polynomial-Time Approximation Scheme)*

For a minimization problem Π, a FPTAS algorithm A takes an instance I and error bound $\epsilon > 0$, and returns a solution s such that $f(s,I) \leq (1+\epsilon) \cdot OPT(I)$. A runs in polynomial time in terms of both $|I|$ and ϵ (i.e. ϵ can not appear in exponent).

Example: $O(\frac{n^2}{\epsilon^2})$ vs. $O(n^{\frac{1}{\epsilon}})$

Some problems like Knapsack, Euclidean TSP, and some scheduling problems belong to class PTAS. Problems like MAX-SAT, Vertex-Cover, are much harder and don't admit a PTAS. These problems belong to a class called APX-hard, which is the class of problems that have a constant approximation but don't have a PTAS.

Definition 1.7 *A problem belongs to the class APX if it has a c-approximation algorithm for some constant $c > 1$.*

Theorem 1.8 *(Arora, Lund, Motwani, Sudan, Szegedy, 1992): There is no $PTAS$ for APX-hard problems unless $P = NP$.*

Some problems are much harder to approximate, in that not only they don't have a PTAS, but also they don't have a constant or even logarithmic factor approximation algorithms.

Theorem 1.9 *(Hastad 1996): There is no $O(n^{1-\epsilon})$-approximation algorithm for Max-Clique for any $\epsilon > 0$ unless $NP \subseteq RP$ (RP, short for Randomized Polynomial Time).*

To appreciate the significance of this theorem note that the naive algorithm which picks only one vertex of the input graph and returns it as the maximum clique has approximation factor $O(n)$ since the size of the largest clique in any graph is at most $O(n)$.