COMPSCI 330: Design and Analysis of Algorithms November 25, 2014

Lecture # 25

Lecturer: Debmalya Panigrahi *Scribe: Roger Zou*

1 Overview

We introduce the Set Cover Problem, and describe a Greedy $\log n$-approximation algorithm for the problem. We then introduce Polynomial Time Approximation Schemes (PTAS), and apply it to 0-1 Knapsack problem by convert from its pseudo-polynomial dynamic programming algorithm to a Fully Polynomial Time Approximation Scheme (FPTAS).

2 Greedy Set Cover

2.1 Introduction

The Set Cover Problem: Given universe X, where $|X| = n$, and sets $S_1, \ldots, S_m$, where $S_i \subseteq X$, $\forall i$. The goal is to select the minimum number of subsets $\mathscr{T}$ such that $\bigcup_{S_i \in \mathscr{T}} S_i$.

Note that the Vertex Cover Problem reduces to Set Cover: Given $G = (V, E)$, let edge $e_i \in E$ correspond to $x_i \in X$, and a set S_i corresponds to all edges adjacent to vertex $v_i \in V$.

2.2 Algorithm

Repeat until all elements in X are covered: Add subset covering the maximum number of uncovered elements to $\mathscr{T}$.

2.3 Analysis

Let OPT be the optimal (NP-Hard) solution to Set Cover. Let t^* be the number of sets in OPT.
Let n_1 be the size of the first set selected by the greedy algorithm. First, note that

$$n_1 \geq \frac{n}{t^*}$$

This is true because n_1 is the largest set in OPT by construction of the algorithm, and $\frac{n}{t^*}$ is the average size of the set. Then the size of the largest set must be at least as large as the size of the average set.

Using the same logic, let n_i be the size of the next largest set of vertices still uncovered (the i-th set selected by the algorithm). Then

$$n_i \geq \frac{n - \sum_{j<i} n_j}{t^*}$$

Let k be the number of sets picked by the algorithm. To show this is a $\log n$-approximation algorithm, we need to show that $k \in O(\log n)$. (It is not known if an algorithm can do better than $O(\log n)$).

First, observe that

$$\begin{aligned}
t^* \frac{n_1}{n} &= t^* \left(\frac{1}{n} + \ldots + \frac{1}{n} \right) \\
&\leq t^* \left(\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{n-n_1+1} \right) \\
t^* \frac{n_2}{n-n_1} &= t^* \left(\frac{1}{n-n_1} + \ldots + \frac{1}{n-n_1} \right) \\
&\leq t^* \left(\frac{1}{n-n_1} + \frac{1}{n-n_1-1} + \ldots + \frac{1}{n-n_1-n_2-1} \right)
\end{aligned}$$

The pattern continues for all n_i.
Then

$$\begin{aligned}
k &= 1 + 1 + \ldots + 1 \quad \text{(k times)} \\
&= \sum_{i=1}^{k} n_i \frac{1}{n_i} \\
&\leq \sum_{i=1}^{k} n_i \frac{t^*}{n - \sum_{j<i} n_j} \\
&\leq t^* \left(\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{n-n_1+1} + \frac{1}{n-n_1} + \ldots + \frac{1}{n-n_1-n_2} + \ldots + 1 \right) \\
&= t^* H_n \\
&= t^* O(\log n)
\end{aligned}$$

where recall the harmonic series H_n is $O(\log n)$.

An alternate proof can use the charging method we learned earlier in class.

3 Polynomial Time Approximation Schemes (PTAS)

3.1 Introduction

A PTAS algorithm requires that, for an arbitrarily small ε...
For a minimization problem: Given any $\varepsilon > 0$, PTAS algorithm $\leq (1+\varepsilon)$ OPT
For a maximization problem: Given any $\varepsilon > 0$, PTAS algorithm $\geq (1-\varepsilon)$ OPT

Fully Polynomial Time Approximation Schemes (FPTAS): PTAS with time complexity poly($\frac{1}{\varepsilon}, n$).

Any problem with a FPTAS solution is **weakly NP-Hard**. Otherwise, it's **strongly NP-Hard**.

3.2 0-1 Knapsack

Recall the NP-Hard 0-1 Knapsack problem. We have items $a_1, a_2, \ldots a_n$, each with a size s_i and profit p_i, and a knapsack of capacity B. The goal is to find subset of items X such that $\sum_{i:a_i \in X} s_i \leq B$, and $sum_{i:a_i \in X} p_i$ is maximized. Here we will show that the pseudo-polynomial dynamic programming solution can be converted to a FPTAS. Thus in some sense this problem is one of the ”easier” NP-Hard problems.

Recall the DP solution. Let $A[i,p]$ be the minimum total size of a set of times having total profit P among items $a_1,\dots,a_i$. Then

$$A[i+1,p] = \begin{cases} A[i,p] & \text{if } p_{i+1} \geq p \\ \min\left\{A[i,p], A[i,p-p_{i+1}+s_{i+1}]\right\} & \text{if } p_{i+1} \leq p \end{cases}$$

Since the maximum possible profit is nP, the DP matrix is n^2P. Each entry of the matrix is filled in constant time. Thus the running time is $O(n^2P)$, which is pseudo-polynomial because of P.

3.3 Scaling

Scaling is the technique we'll use to convert the DP algorithm to a FPTAS.

1. Select $k = \frac{\varepsilon P}{n}$, the scaling parameter.
2. $p_i^* = \lfloor \frac{p_i}{k} \rfloor$.
3. Run the DP algorithm on P^*.
4. Scale all profits back by k.

The running time is $O(n^2 * \frac{n}{\varepsilon}) = O(n^3\frac{1}{\varepsilon})$.

3.4 Analysis

The claim is that the solution to P^* lower bounds the optimal solution to P by factor $1-\varepsilon$. returns $k*$ OPT(P^*). From the scaling, any p_i gets converted to $\lfloor \frac{p_i}{k} \rfloor k$. Thus

$$\begin{aligned} p_i &\leq \left\lfloor \frac{p_i}{k} \right\rfloor k + k \\ &= p_i^* k + k \end{aligned}$$

So

$$\text{OPT}(P) \leq k * \text{OPT}(P^*) + kn$$

Then

$$\begin{aligned} \text{Solution returned by algorithm} &= k * \text{OPT}(P^*) \\ &\geq \text{OPT}(P) - nk \\ &= \text{OPT}(P) - \varepsilon P \\ &\geq \text{OPT}(P) - \varepsilon\text{OPT}(P) \\ &= (1-\varepsilon)\text{OPT}(P) \end{aligned}$$

So the scaled DP algorithm is a FPTAS.