CSC 2421 - Approximation Algorithms

Lecture #1: 7 January 2003

Lecturer: A. Borodin

Scribe Notes by: S. Alam and Y. Ahmed

1 Course Overview

This is a research oriented course about approximation algorithms. The course is divided into two parts.

- Part A : Positive Results (Mostly presented by A. Borodin)
- Part B : Theory and Negative Results (Mostly presented by C. Rackoff)

Part A consists of approximation ratios for some fundamental optimization problems and general algorithmic strategies. The problem domain list includes scheduling problems, graph theory, counting problems, covering and packing, geometric problems. The algorithmic strategies are greedy, dynamic programming, local search, linear programming (LP) and rounding, LP primal dual, semi-definite programmming, sampling, markov chains.

Part B deals with the results pertaining to the theory of approximation algorithms which includes MAXSNP and MAXSNP completeness, the classes PLS and GLO, PCP proofs and hardness of approximation.

In the rest of today's lecture we start with a "simple" NP-optimization problem: Vertex Cover. Then we discuss about some basic definitions of NP-optimization problems and related matters. Finally, we finish by introducing another NP-Optimization problem Set Cover.

2 Vertex Cover Problem (VC)

Definition

- **Input:** An undirected graph $G = (V, E)$
- **Goal:** Find a set $U \subseteq V$ of minimum size $|U|$ such that $\forall (u, v) \in E$, we have either $u \in U$ or $v \in U$.

The "obvious" Greedy algorithm for this problem is:

$U \leftarrow \phi$

While $E \neq \phi$

Choose vertex v of maximum degree of remaining graph.

$U \leftarrow U \cup \{v\}$

delete v and all adjacent edges of v

This is an H_Δ-approximation algorithm where Δ is the maximum vertex degree in the graph. (Here H_n denotes the n-th harmonic number which is $1 + 1/2 + 1/3 + ... + 1/n$ and $\ln n < H_n < \ln n + 1$)

2.1 A greedy algorithm using Maximal Matching

Let us define maximal matching first. Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a matching if no two edges of M share an endpoint. A maximal matching is simply a matching that cannot be extended to a larger matching. The following algorithm uses a maximal matching to find a vertex cover.

$U \leftarrow \phi$

While $E \neq \phi$

let (u, v) be an arbitrary edge of E

$U \leftarrow U \cup \{u, v\}$

remove from E every edge incident on either u or v

return U

Claim 1: U is a vertex covering.

Claim 2: $|U| \leq 2 \cdot \mathrm{OPT}_{VC}$ i.e this is a 2-approximate algorithm for the cardinality VC problem.

Proof: let M be a maximal matching for G. So $|M| \leq \mathrm{OPT}_{VC}$ because for each edge in M, at least one end point must be in the optimal vertex cover. The algorithm returns a vertex cover of size $|U| = 2|M| \leq 2 \cdot \mathrm{OPT}_{VC}$.

This 2-approximation algorithm is close to the best known polynomial time algorithm and many believe $2 - o(1)$ might be best possible approximation ratio if P≠NP.

Claim 3: The analysis presented above is tight.

Consider the infinite family of instances given by the complete bipartite graphs $K_{n,n}$ (see Figure 1). When run on $K_{n,n}$, the above algorithm will pick all $2n$ vertices, whereas picking one side of the partition gives a cover of size n.

2.2 Weighted Vertex Cover Problem

In the weighted version a weight is assigned to each vertex. Formally WVC is :

- **Input:**
  - An undirected graph $G = (V, E)$
  - Weights $w_i \geq 0 \;\; \forall i \in V$

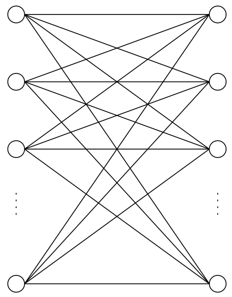

Figure 1: Bipartite Graph $K_{n,n}$

- **Goal:** Find a vertex cover C which minimizes $\sum_{i \in C} w_i$.

2.3 The Weighted Vertex Cover approximation results

These algorithms by BarYehuda and Even and by Monien and Speckenmeyer achieve a ratio of $\left(2 - \frac{\ln \ln n}{2 \ln n}\right)$, where n is the number of vertices in the graph. In graphs with maximal degree Δ, Hochbaum shows how to approximate the vertex cover problem within a factor of $\left(2 - \frac{2}{\Delta}\right)$. Halldorsson and Radhakrishnan improve this and give a $\left(2 - \frac{log\Delta + O(1)}{\Delta}\right)$ approximation algorithm for the problem. Halperin improves this result and give a $\left(2 - (1 - o(1))\frac{2 \ln \ln \Delta}{\ln \Delta}\right)$ approximation algorithm. Johan Hastad has proved that a performance ratio less than $\frac{7}{6}$ is not possible unless P=NP. Dinun and Safra found the best lower bound which is $10\sqrt{5} - 21 \approx 1.36067 > \frac{7}{6}$.

3 Some basic definitions

An NP-optimization problem Π consists of:

- **Valid instances:** A set of valid instances, D_Π, recognizable in polynomial time e.g $D_\Pi \in P$. The size of an instance $I \in D_\Pi$, denoted by $|I|$, is defined as the number of bits to write I under the assumption that all numbers occurring in the instances are written in binary.
- **Feasible solution:** Each instance $I \in D_\Pi$ has a set of feasible solution, $S_\Pi(I)$. We require that $S_\Pi(I) \neq \phi$ and that every solution $s \in S_\Pi(I)$ is of length polynomially bounded by I; i.e. $|s| \leq |I|^{o(1)}$
- **Objective function:** A polynomial time computable objective function, obj_Π or f_Π assigns a nonnegative rational number to each pair (I, s) where I is an instance and s is a feasible solution for I; i.e. $f_\Pi(I, s) \in Q^{\geq 0}$ where $s \in S_\Pi(I)$.
- Π is specified as a *minimization* or *maximization* problem

The vertex cover problem can be seen as a NP-optimization problem. If $|\{f_\Pi(I, s) \,|\, |I| \leq n\}| \leq n^{O(1)}$ then Π is called a polynomially bounded optimization function e.g. the cardinality vertex cover is polynomially bounded.

3.1 Decision Problem associated with Π

Every NP-optimization problem Π can be naturally associated with a decision problem by giving a bound on the optimal solution. Decision version of Π is

- **Input:** A pair (I, B) where I is an instance of Π and B is a rational number.
- **Goal:** Determine
  - If $\exists s \in S_\Pi(I) : f_\Pi(I, s) \leq B$ when Π is a minimization problem.
  - If $\exists s \in S_\Pi(I) : f_\Pi(I, s) \geq B$ when Π is a maximization problem.

The answer to the decision problem is "yes" if the above condition holds, otherwise it is "no". We say that an NP-optimization problem is NP-hard if the associated decision problem is NP-hard.

3.2 Factor δ approximation algorithm

Let Π be a minimization (maximization) problem, and let δ be a function, $\delta : \mathbf{Z}^+ \to \mathbf{Q}^+$, with $\delta \geq 1$ ($\delta \leq 1$). An algorithm $\mathcal{A}$ is said to be a *factor δ approximation algorithm for Π* if $\forall I$, $\mathcal{A}$ produces $s \in S_\Pi(I) : f_\Pi(I, s) \leq \delta(|I|) \cdot \mathrm{OPT}(I)$ (respectively, $f_\Pi(I, s) \geq \delta(|I|) \cdot \mathrm{OPT}(I)$), and the running time of $\mathcal{A}$ is bounded by a fixed polynomial in $|I|$. Clearly, the closer δ is to 1, the better is the approximation algorithm.

Notation: For simplicity, we sometimes abuse the notation and let $\mathcal{A}_\Pi(I)$ be either the value or the solution produced by $\mathcal{A}$.

There are two conventions for maximization problem:

- $\mathrm{OPT}(I) \leq \delta(|I|) \cdot \mathcal{A}_\Pi(I)$ where $\delta \geq 1$
- $\mathcal{A}_\Pi(I) \geq \delta(|I|) \cdot \mathrm{OPT}(I)$ where $\delta \leq 1$

Usually, we have $\delta(\#I)$ where $\#I$ stands for some natural parameters of I; e.g. degree of a graph etc. But we will just say $|I|$ for simplicity.

3.3 Asymptotic δ-approximation algorithm

$$\lim_{|I| \to \infty} SUP \frac{\mathcal{A}_\Pi(I)}{\mathrm{OPT}(I)} \leq \delta(|I|)$$

Equivalently, assuming $\mathrm{OPT}(I) \to \infty$ as $|I| \to \infty$

$$\mathcal{A}_\Pi(I) \leq \delta(|I|) \cdot \mathrm{OPT}_\Pi(I) + o(\mathrm{OPT}_\Pi(I))$$

3.4 Factor δ randomized approximation algorithm

A is a *factor δ randomized approximation algorithm* for Π if, $\forall I$ $\mathcal{A}$ produces a feasible solution s for I such that

- $\mathbf{Pr}\left[f_\Pi(I,s) \le \delta(|I|)\cdot \mathrm{OPT}(I)\right] \ge \frac{1}{2}$ where Π is a minimization problem.
- $\mathbf{Pr}\left[f_\Pi(I,s) \ge \delta(|I|)\cdot \mathrm{OPT}(I)\right] \ge \frac{1}{2}$ where Π is a maximization problem.

One can run $\mathcal{A}$ repeatedly t-times to get probability $\ge 1 - \frac{1}{2^t}$.

3.5 Polynomial Time Approximation Scheme(PTAS) for Π

A Uniform PTAS for a problem Π is an algorithm $\mathcal{A}$ that takes an input instance I of a minimization problem Π and an error bound ϵ and returns a solution $\mathcal{A}(\mathcal{I},\epsilon)$ s.t.

$$\mathcal{A}(I,\epsilon) \le (1+\epsilon)\cdot \mathrm{OPT}(I)$$

The running time is polynomial in $|I|$ for each fixed ϵ.
In case of non-uniform PTAS we have a collection of algorithms $\{A_\epsilon\}$ because we need a different algorithm A_ϵ for each $\epsilon \ge 0$.

3.6 Fully Polynomial Time Approximation Scheme(FPTAS) for Π

A FPTAS for a problem Π is an algorithm $\mathcal{A}$ that takes as input instance I of Π and an error bound ϵ and returns a solution $\mathcal{A}(\mathcal{I},\epsilon)$ s.t.

$$\mathcal{A}(I,\epsilon) \le (1+\epsilon)\cdot \mathrm{OPT}(I)$$

The running time of $\mathcal{A}(I,\epsilon)$ is polynomial in both $|I|$ and $\frac{1}{\epsilon}$.

4 Two more curiosities

Two problems that are worth mentioning in this context are

- **Bin Packing:**
  - **Input:** n items with sizes $a_1, ..., a_n \in (0,1]$.
  - **Goal:** Find a packing in unit sized bins that minimizes the number of bins used.

  It is NP-hard to determine whether 2-bin suffices (in this case, it is essentially a reduction from the partition problem).

  For any $\epsilon > 0$, there is no (3/2 - ϵ) approximation algorithm for bin packing problem if P$\neq$NP. But there exists some algorithm $\mathcal{A}_\Pi$ for Bin packing problem s.t.

$$\mathcal{A}_\Pi(I) \le 1\cdot \mathrm{OPT}(I) + o(\mathrm{OPT}(I)) \;\text{ where }\; o(\mathrm{OPT}(I)) \approx log^2\mathrm{OPT}(I)$$

  That is, $\mathcal{A}_\Pi$ is an *asymptotic* 1-approximation algorithm.

- **Problems which are trivially equivalent w.r.t decision problem:** Such problems can have a dramatically different approximation ratios (assuming $P \neq NP$). Vertex Cover Problem and Independent Set Decision Problem are closely equivalent i.e $U \subseteq V$ is a V.C iff $(V - U)$ is an independent set. Hence

  $$\text{OPT}_{V.C.}(I) = n - \text{OPT}_{Max-Independent-Set}(I)$$

  .

  Max Independent Set is NP-hard to approximate within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless NP=ZPP (ZPP is short for *Zero Error Probabilistic Polynomial Time*).

5 The Set Cover Problem

Definition

- **Input**
  - A Universe $U = \{e_1, e_2, ..., e_n\}$.
  - A collection of subsets of U, $\mathcal{S} = \{S_1, ..., S_k\}$.
  - A cost function $c : \mathcal{S} \to \mathbf{Q}^{\geq 0}$.
- **Goal:** Find a minimum cost subcollection of $\mathcal{S}$ that covers all elements of U.

Frequency: The *frequency* of an element is the number of sets it is in. Let f be the frequency of the most frequent element. More formally

$$f = \max_{e} |\{i : e \in S_i\}|$$

The various approximation algorithms for set cover achieve roughly one of the two factors: H_n or f.
Claim: The set cover problem with $f = 2$ is essentially the vertex cover problem.