CS 221: Computational Complexity **Prof. Salil Vadhan**

Lecture Notes 23

April 21, 2010 **Scribe: Rebecca A. Resnick**

Agenda

- Inapproximability
  - Max Ind. Set
  - Survey
- Alegbraic Complexity
  - Model
  - Complexity classes
  - Completeness of Det and Perm

1 Recall

- There exists a poly-time 2-approximation algorithm for MIN-VC
- There exists $\varepsilon > 0$ such that $\text{Gap}_{\frac{2}{3},\frac{2+\varepsilon}{3}}$Min-VC is **NP**-hard.

  This implies that there does not exists a poly-time $(1+\varepsilon')$ approximation algorithm for Min-VC with $\varepsilon' < \varepsilon/2$ unless $\mathbf{P} = \mathbf{NP}$. So Min-VC has a poly-time approximation, but not an arbitrarily good one.

2 Inapproximability

2.1 MAX-IS

Definition 1 Max-IS*: Given a graph G, find an independent set of maximum size.*

Theorem 2 *(Assuming PCP theorem) For every constant $\rho > 0$, there exists no ρ-approximation for MAX-IS unless* $\mathbf{P} = \mathbf{NP}$

Proof:

Lemma 3 $\text{Gap}_{1-a,1-b}$Min-VC $\leq_l$ $\text{Gap}_{a,b}$Max-IS.

Here $\text{Gap}_{a,b}$Max-IS, for $a > b$ is the promise problem where YES instances are graphs with an independent set of size at least an and NO instances are graphs in which every independent set has size at most bn.

Proof: The reduction is the identity mapping, and the correctness follows from the fact that a set S is a vertex cover iff its complement is an independent set. ■

Lemma 4 $\textsc{Gap}_{a,b}\textsc{Max-IS} \le \textsc{Gap}_{a^k,b^k}\textsc{Max-IS}$.

Note: if $a > b$, then $b^k/a^k \to 0$ as $n \to \infty$

Proof: Given $G = (V, E)$, map $(V, E) \mapsto G_k = (V^k, E_k)$, where

$$E_k = \{(u_1, \ldots, u_k), (v_1, \ldots, v_k) : \exists i (u_i, v_i) \in E\}.$$

If S is an i.s. in G, then S^k is an i.s. in G_k, so MAX-IS$(G_k) \ge$ MAX-IS$(G)^k$. We want to show that the converse holds:

Let $T \subseteq V^k$ be an i.s. in G_k. Then the coordinate-wise projections $\pi_1(T), \ldots, \pi_k(T)$ are all ind. sets in G (if you have 2 coordinates which are connected in G, there is an edge between them in G_k). Then

$$T \subseteq \pi_1(T) \times \ldots \times \pi_k(T),$$

which implies that

$$|T| \le \text{MAX-IS}(G^k).$$

Hence, MAX-IS$(G_k) =$ MAX-IS$(G)^k$ ■

Note: This says nothing about VC. In VC we would have $\frac{1-b^k}{1-a^k} \to 1$, so this method of "amplification" gets worse and worse in VC. Moral of the story: switching from maximization to minimization is *not* equivalent for approximation. ■

2.2 Survey

Below $\varepsilon > 0$ denotes an arbitrarily small constant.

Problem	Best known approximation algorithm	**NP**-hard
Euclidean TSP	$(1+\varepsilon)$-approx	
Max-3SAT	7/8-approx	$(7/8+\varepsilon)$-approx
Min-SetCover	$\ln n$-approx	$(1-\varepsilon)\ln n$-approx
Max-IS	$(\text{polylog}(n)/n)$-approx	$(1/n^{1-\varepsilon})$-approx

Note: The **NP**-hardness of the above problems is proven using PCP optimized for each problem. You begin with a PCP problem and do clever amplifications, compositions.

Here are some problems where we don't have tight **NP**-hardness results:

Problem	Algorithm	**NP**-hard	UG-hard
Min-VC	2-approx	$\approx 1.36\ldots$-approx	$(2-\varepsilon)$-approx
Max-Cut	$.878\ldots$-approx (semi-definite programming)	$17/16 - 3 \approx .94\ldots$-approx	$.878\ldots$-approx
Shortest Vector	$\approx 2^{n/\log n}$-approx (looks bad, but really useful)	$2^{\log^{1-\varepsilon} n}$-approx	

Where Max-Cut is the problem of partitioning a graph so that a single cut will sever as many edges as possible, and SHORTEST VECTOR is the problem of finding the approximate length of the shortest vector in a lattice graph.

UG = "unique games": this is an approximation problem/PCP variant that is conjectured to be hard. An equivalent problem: given an (inhomogeneous) system of linear equations mod q, with 2 variables per equation, where $q = q(\varepsilon)$ is a large constant, distinguish $(1 - \varepsilon)$-satisfiable from ε-satisfiable. If this is hard, then our understanding of alot of approximation problems gets resolved (as illustrated by the examples of MIN-VC and MAX-CUT above). As a result, this conjecture is currently the subject of intense study. This study has also uncovered interesting connections with mathematical questions in metric geometry and discrete Fourier analysis.

3 Algebraic Complexity

Question: How many arithmetic operations are needed to compute various polynomials of interest?

3.1 Model

Look at algebraic circuits (and formulas), $C(x_1, \ldots, x_n)$ over a fixed field $\mathbb{F}$

- inputs: $x_1, \ldots, x_n$ and constants from $\mathbb{F}$.
- gates: $+, \cdot$.

(Fact: $\div$ doesn't help much.) We view algebraic circuits as computing *formal* polynomials over the field $\mathbb{F}$. These can be evaluated at points in $\mathbb{F}$ (by substituting for the variables x_i), but are not necessarily determined only by the function they compute. (For example, x^2 and x are different polynomials over $\mathbb{Z}_2$, even though they compute the same function.)

3.2 Complexity Measures

- size = #gates (including inputs)
- non-scalar complexity/#"essential" operations = # multiplications *not* by a constant.

  Motivation: this measure seeks to count the most expensive operations
- depths (as in boolean circuits): longest path from input to output, measures parallelism.

3.3 Examples

- MATRIX MULTIPLICATION:

$$(X_{ij})(Y_{ij}) \to (Z_{ij} = \Sigma_k X_{ik} Y_{kj})$$

  This sends $2n^2$ variables to n^2 output polynomials.

  Naive algorithm: $O(n^3)$

  Best known algorithm: $O(n^{2.37})$

  Best lower bound: $\approx 3n^2$

- Discrete Fourier Transform: $\mathbb{F} = \mathbb{C}$

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \mapsto \begin{pmatrix} & & \\ & \omega^{ij} & \\ & & \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}. \tag{1}$$

  Where $\omega =$ the primitive nth root of unity.

  Naive algorithm: $O(n^2)$

  Fast Fourier Transform (FFT): $O(n \log n)$

  Best lower bound: no super linear lower bounds are known.

- Determinant:

  Naive algorithm: $O(n \cdot n!)$

  Gaussian elimination: $O(n^3)$

  Best known: $O(n^{\ 2.37}$

- Permanent:

  Naive algorithm: $O(n \cdot n!)$

  Best known algorithm: $O(2^n)$

Definition 5 *Fix a field $\mathbb{F}$. A sequence $(p_n(x_1 \ldots, x_n))_{n\in\mathbb{N}}$ of polynomials over $\mathbb{F}$ is in* **AlgP/poly** *(also called* **VP** *for "Valiant's* **P**"*) if there exist polynomials $d(n), s(n)$ such that for all n*

1. $\deg(p_n) \leq d(n)$
2. *p_n is computable by an arithmetic circuit of size at most $s(n)$.*

Why bound degree?

- $\deg \leq 2^{\text{size}}$ in any arithmetic circuit
- most functions of interest have low polynomially bounded degree
- it is useful in results

Definition 6 *$(p_n(x_1 \ldots, x_n))_{n\in\mathbb{N}}$ is in* **AlgNP/poly** *(a.k.a.* **VNP***) if there exists a sequence of polynomials $(q_n(x_1 \ldots, x_n))_{n\in\mathbb{N}}$ in* **AlgP/poly** *and a polynomial $t(n)$ such that for all n,*

$$p_n(x_1, \ldots, x_n) = \sum_{e_{n+1},\ldots e_{t(n)} \in \{0,1\}} q_{t(n)}(x_1, \ldots, x_n, e_{n+1}, \ldots, e_{t(n)})$$

3.4 Next time

Theorem 7
- Det *is complete for* **AlgP/poly**.
- Perm *is complete for* **AlgNP/poly**.

Hence, **AlgP/poly** = **AlgNP/poly** $\iff$ Perm is a "projection" of the determinant.