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

Lecture Notes 15

March 24, 2010 **Scribe: Mike Ruberry**

1 Recap and Introduction

Recall the following definitions for a function $f : \{0,1\}^* \to \mathbb{R}^{\geq 0}$, and $\alpha > 1$:

α-approximation algorithm for f : $\forall x\ f(x) \leq A(x) \leq \alpha \cdot f(x)$

randomized α-approximation $\Pr[f(x) \leq A(x) \leq \alpha \cdot f(x)] \geq \frac{2}{3}$

approximation scheme for all $\varepsilon > 0$, we have $\forall x\ f(x) \leq A(x, \varepsilon) \leq (1+\varepsilon)f(x)$, and A runs in time $p_\varepsilon(|x|)$ for a polynomial p_ε (that can have an arbitrarily bad dependence on ε

fully polynomial approximation scheme the above $A(x, \epsilon)$ runs in time $\text{poly}(|x|, 1/\varepsilon)$; that is, time polynomial in the input and the inverse of the approximation factor, ϵ

One of the main points of today's lecture is that approximate counting tends to be much easier than exact counting.

2 Relation to promise problems

Having an α-approximation algorithm $A(x)$ is equivalent to having an algorithm which decides the following promise problem:

$$\text{Gap}_\alpha f \quad \begin{array}{l} \textbf{YES} = \{(x,t) : f(x) \geq t\} \\ \textbf{NO} = \{(x,t) : f(x) < t/\alpha\} \end{array}$$

or alternatively:

$$\text{Gap}_\alpha f \quad \begin{array}{l} \textbf{YES} = \{(x,t) : f(x) \geq \alpha t\} \\ \textbf{NO} = \{(x,t) : f(x) < t\} \end{array}$$

What's important is the gap between inequalities which permits us to disambiguate.

A subtle but important point is the distinction between $\text{Gap}_{1+\epsilon}$ #CIRCUIT-SAT and $\epsilon-$APPROX-CIRCUIT-ACCEPT-PROBABILITY. The latter problem has a natural **PrBPP**-complete analogue:

$$\textbf{YES} = \{(c,p) : Pr[C(x) = 1] \geq p + \epsilon\}$$
$$\textbf{NO} = \{(c,p) : Pr[C(x) = 1] \leq p\}$$

This latter approximation is within $\pm\epsilon \cdot 2^n$, where n is the number of inputs to the circuit, C, whereas the former is within $\pm\epsilon \cdot k$, where k is the number of satisfying assignments. On instances with few satisfying assingments this latter approximation must be increasingly precise and in particular can disambiguate between zero and one satisfying assignments, allowing it to solve problems like circuit satisfiability and thus making it **NP**-hard.

From this discussion we conclude **approximate counting is at least as hard as decision**.

3 Examples

There are many #**P**-complete problems with fully polynomial randomized approximation schemes:

#DNF while for exact counting DNF is the same as CNF, approximate counting is easier with DNF

#Perfect Matchings (in bipartite graphs), and more generally the nonnegative Permanent.

and more from statistical physics

These often work by using Markov chain Monte Carlo is used to generate solutions (almost) uniformly at random (which is broadly equivalent to approximate counting).

Uniform Sampling let M be a polynomial time verifier with solution length $p(n)$. A *fully polynomial almost-uniform sampler* for M is a probabilistic algorithm, S, such that $\forall x \in \{0,1\}^*, \epsilon > 0$, the random variable $S(x,\epsilon)$ is ϵ-"close" to the uniform distribution on the set of solutions for x, namely $\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}$, and $A(x,\epsilon)$ runs in time poly$(|x|, 1/\epsilon)$.

Closeness of random variables Two random variables X, Y are "ϵ-close" if for every set T of possible outcomes $|\Pr[X \in T] - \Pr[Y \in T\| \leq \epsilon$.

Our description will be robust to many notions of "distance," however.

4 Relation of almost uniform samplers and approximate counting

Theorem 1 *For "downward self-reducible" verifiers (defined only by example); #M (counting solutions to M) has a fully polynomial randomized approximation scheme if and only if M has a fully polynomial almost uniform sampler.*

Proof: We will prove that a having a sampler provides an approximation counter for the number of matchings (not just perfect matchings) in a graph. The other half of the proof is left as an exercise for problem set 4.

Assume we have a fully polynomial sampler, S. Iur approximating counting algorithm $A(G,\epsilon)$ works as follows:

1. if G has no edges, output 1; otherwise let $e = (u, v)$ be an edge in G
2. run sampler $S(G, \delta), O(n/\delta^2)$ times to estimate $\hat{p}$, the fraction p of matchinges in G that contain e (w.p. $1 - 2^{-n}, |\hat{p} - p| \leq 2\delta$ by a Chernoff bound. The first δ is from the bound, the second from the sampler.)
3. if $\hat{p} < 1/2$, recursively estimate the number of matchings in $G \setminus e$ (which exactly equals the number of matchings in G without edge e) and output $(A(G \setminus e, \delta)/(1 - \hat{p})$. The numerator represents the recursion, while the denominator is the ratio to the total number of matchings.
4. alternatively, if $\hat{p} > 1/2$ then we recursively estimate the number of matchings $G \setminus \{u, v\}$ (both vertices and all connected edges removed from G), which is exactly equals the number of matchings where (u, v) are matched using e. We then output $A(G \setminus \{u, v\}, \delta)/\hat{p}$.

The two cases permit a better approximation: if p is close to zero then we would have a bad relative approximation to p, but an excellent approximation to $1 - p$.

We now will show by induction that $A(G, \epsilon) \in (1 \pm O(\delta))^{\#edges\ in\ G} \cdot \#MATCH(G)$. Let $\delta = \frac{\epsilon}{cm}$, which will give relative error $\leq 1 + \epsilon$.

We now consider the 4th point above:

$$p \leq \hat{p} + 2\delta \leq \hat{p} \cdot (1 + 4\delta) \text{ where } (\hat{p} > \tfrac{1}{2})$$

$$p \geq \hat{p} - 2\delta \geq \hat{p}(1 - 2\delta)$$

and by induction $\frac{A(G\setminus\{u,v\},\epsilon)}{\hat{p}} = \frac{(1\pm O(\delta))^{m-1}\cdot\#\text{matchings}(G\setminus\{u,v\})}{(1\pm O(\delta))\cdot p} = (1 \pm O(\delta))^m \cdot \#\text{matchings}(G)$. ■

From the example, we can see that "downward self reducible" means that we can fix a constant amount of the witness (e.g. whether edge e is present or not in the matching) and obtain a strictly smaller instance of the same problem (e.g. $G \setminus e$ or $G \setminus \{u, v\}$).

5 Decision and counting

We know that approximate counting is at least as hard as decision—are there cases where decision is easy and approximate counting is hard? **Yes!**

Theorem 2 *Approximate # CYCLES in a graph is **NP**-hard.*

Proof: Deciding whether a graph has a cycle is in **P**, of course. We show hardness by reduction from Hamiltonian Cycle: For every constant $\alpha > 0$, Hamiltonian Cycle$\leq_l$ GAP$_\alpha$ CYCLES. We want to map our graph, G to a new graph, (G', t) where if G has a Hamiltonian cycle then the number of cycles in $G' \geq t$. Alternatively, if G does not have a Hamiltonian cycle then the number of cycles in G' is $< t/\alpha$.

We'll show this for directed graphs, but the same proof applies in the undirected case, too. Intuitively, we make big cycles give us many cycles, small cycles fewer. For an edge between (u, v) we construct a series of l "diamonds" such that there are 2^l ways to go from u to v now. If G had a Hamiltonian cycle then G' has $\geq 2^{l \cdot n}$, while if G does not have a Hamiltonian cycle then G' has fewer than $2^{l(n-1)}n!$ We must be careful to pick $l = n^2$ to dominate the $n!$ factor. ■

5.1 How hard can approximate counting be?

Approximate counting (and uniform sampling) can always be done in probabilistic polynomial time with an **NP** oracle: $\forall f \in \#\,\mathbf{P}$, $\textsc{Gap}_\alpha f \in \mathbf{BPP^{NP}} \subseteq \mathbf{\Sigma_3^p} \cap \mathbf{\Pi_3^p}$. We'll see this in a few weeks. This is in sharp contrast to exact counting, which is more powerful than the entire **PH** by Toda's Theorem.

6 Next time: Average case complexity

Goal: understand hardness of problems on "random" inputs.

Let $f : \{0,1\}^* \to \{0,1\}^*$, let D_n be a distribution on $\{0,1\}^n$ for every n, and $\delta : \mathbb{N} \to [0,1]$. Then we say that f is *δ-hard on $D = \{D_n\}$* if $\forall$ PPT A, $\Pr_{x \leftarrow D_n, coins\ of\ A}[A(x) \neq f(x)] > \delta(n)$.

That is, every efficient algorithm must err with probability greater than $\delta(n)$ on random inputs chosen according e.g. the expected run time under D. People study other ways to model average-case hardness, such as considering algorithms A that are always correct but considering their expected running time under D_n, but we will focus on this basic definition for our brief study of the subject.

6.1 Motivations

- model "real life" instances better than worst-case complexity? (challenge is to find distributions that are a good model)
- cryptography, and some other areas like derandomization (here we want to generate hard instances in order to use their hardness for useful things like encrypting data or producing "pseudorandom" bits")

6.2 Examples of problems conjectured to be average-case hard

- $f(x)$ = prime factorization of x, where D_n is the product of two random $n/2$-bit primes
- f = SAT, D_n is a random Δ_n 3-clauses, where n is the number of variables and Δ is a fixed constant (the satisfiability threshold).