CS221: Computational Complexity	Prof. Salil Vadhan
Lecture 26: Average Case Complexity, Random Self-Reducibility of the Permanent	
11/25	*Scribe: Alexander Sheh*

Contents

1 Average-Case Complexity **1**

2 Random Self-Reducibility of the Permanent **2**

1 Average-Case Complexity

So far, we have been studying worst case complexity, i.e. maximum complexity over all inputs. But even if $f \notin$ P, "hard instances" may be very rare. How can we study the complexity of function f on "typical instances"? Average-case complexity is one approach: it studies the complexity of problems on *random* instances. Formally, a *distributional problem* is a pair (f, μ) where $f : \{0,1\}^* \to \{0,1\}^*$ and where $\mu = \{\mu_n\}$ is a sequence of distributions μ_n on inputs of length n.

We can measure for algorithm A:
1) Error probability $\epsilon(n) = \Pr_{x \xleftarrow{R} \mu_n}[A(x) \neq f(x)]$. Require A to be efficient (polynomial time).
2) Expected running time $T(n) = \mathrm{E}_{x \xleftarrow{R} \mu_n}[\textit{time taken by A on x}]$. Require A to always be correct. (This is somewhat analogous to what's given by the almost everywhere hierarchy theorems).
3) Both 1 and 2.
We'll focus on measure 1.

Note: choice of distribution μ is important.
e.g. Random 3SAT,
Distribution parameterized by clause density $\triangle$. μ_n = choose $\triangle_n$ clauses uniformly at random from all $2^3 \cdot \binom{n}{3}$ possible clauses $\phi(x_1...x_n) = c_1 \wedge c_2 \wedge \cdots \wedge c_\triangle$.
- $\triangle < 3.76 \Rightarrow Pr[\phi\ satisfiable] \to 1$ as $n \to \infty$.
$A(\phi)$ = "satisfiable" has $\epsilon(n) \to 0$ (Uninteresting!)
- $\triangle > 4.51 \Rightarrow Pr[\phi\ satisfiable] \to 0$. (Also uninteresting)
- $\triangle \simeq 4.2$ "satisfiability threshold" $\to$ possibly contains hard instances.(??)

<u>Motivations</u>

- Understand behavior of algorithms on "real-life" instances
  - This requires modelling the "real-life" distribution on instances. This is often difficult, and tends to be very domain-specific. It has been attempted in some cases, e.g. people have proposed probabilistic models for various kinds of networks (social networks, the WWW, etc.), which could be used as a basis for average-case analysis of algorithms for such networks.

- Cryptography
  - exploits the fact that there are intractable problems
  - need to generate hard instances of problems together with witnesses

- Other theory areas have been influenced by average-case complexity: learning theory, pseudorandomness, etc.

Some Approaches

- Levin's Theory: distributional framework
  - defines "distributional NP", "average P", reduction + completeness for distributional problems
  - unfortunately, very few natural complete problems in this framework (version of bounded halting, tiling problems, matrix algebra, but none that are more combinatorial, i.e. 3SAT, TSP)

- Cryptography
  - define one-way function (o.w.f.) to capture the form of average-case complexity needed, and propose natural candidates (int factoring, RSA)
  - take above as basic assumption. Show $\exists$ o.w.f. $\Leftrightarrow$ many useful cryptographic tasks
  - focus on its own needs, don't worry too much about relation to the rest of complexity theory

- Relate worst-case complexity and average-case complexity
  - this is what we'll study

2 Random Self-Reducibility of the Permanent

We want to relate worst-case complexity to average-case complexity; show that a problem is hard(easy) in the worst-case if and only if it is hard(easy) in the average-case.

Modular Permanent

Input: $(1^k, 1^p, M)$ where p prime $> 3(k+1)$, M is $k \times k$ matrix with entries $\in \mathbb{Z}_p = \{0, ...p-1\}$. $\mathbb{Z}_p$ denotes the integers modulo p.

Output: Perm(M) mod p

We know that the Permanent is $\#\mathbf{P}$-complete. Does "Mod" make it easier? Note that computing Perm(M) mod 2 is easy, because it equals det(M) mod 2 because +/- are same. In fact all Perm(M) mod 2^i are easy, but for more complex reasons.

Proposition 1 Modular Permanent *is* $\#\mathbf{P}$*-hard.*

Proof: We'll show the 0/1 Permanent can be computed in polynomial time with an oracle for ModPerm.
Given a k x k matrix M. Let $p_1, \ldots, p_k$ be the first k primes $> 3(k+1)$.
Compute Perm(M) mod $p_1, \ldots,$ Perm(M) mod p_k.
Combining using Chinese Remainder Theorem, we can compute Perm(M) mod $\prod_{i=1\ldots k} p_i =$ Perm(M), where the last equality follows from $\prod_{i=1\ldots k} p_i > (3(k+1))^k > k! \geq$ Perm(M). ■

This proposition still refers to the worst-case complexity of ModPerm. Now we'll study it's average-case complexity. Let $\mu_{k,p}$ be the uniform distribution on $k \times k$ matrices with entries in $\mathbb{Z}_p$.

Theorem 2 (Lipton) *If there exists a (probabilistic) polytime algorithm A s.t. solving* ModPerm *w.p.* $> 1 - 1/(3(k+1))$ *over* $\mu_{k,p}$*, then there exists a probabilistic polytime algorithm A' solving* ModPerm *on* every *instance w.p.* $> 2/3$ *(over coin tosses of A'). In particular,* ModPerm $\in \mathbf{BPP}$, *so* $\mathbf{P}^{\#P} = \mathbf{BPP}$.

Theorem 3 *Let $\mathbb{F}$ be a finite field,* $|\mathcal{F}| \geq d + 2$. *Given oracle access to function* $f : \mathbb{F}^m \to \mathbb{F}$ *agreeing with some polynomial* $q : \mathbb{F}^m \to \mathbb{F}$ *of degree* $\leq d$ *on a* $> 1 - 1/(3(d+1))$ *fraction of inputs, we can compute q everywhere w.h.p. in time* $\mathrm{poly}(m, \log |\mathcal{F}|, d)$.

Aside on finite fields:

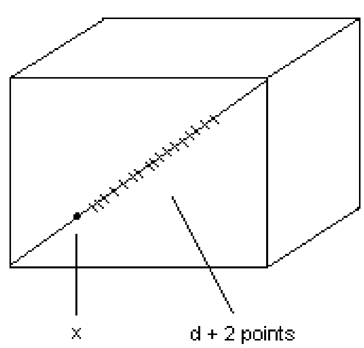
- A field is a set with addition and multiplication s.t. all nonzero elements have multiplicative inverses. (See an algebra text for a full definition.)
- For every prime p, $k \in \mathcal{N}$ (set of natural numbers), $\exists$ finite field of size p^k, denoted GF(p^k) or $\mathbb{F}_{p^k}$ - unique up to isomorphism. For k = 1, GF(p) $\equiv \mathbb{Z}_p$.
- A representation of GF(p^k) can be found in time poly(k, p), or poly$(k, \log p)$ randomized, and computed in time poly$(k, \log p)$. In other words, we can find and compute finite fields efficiently. "Find" refers to finding a prime p of a given size (and if $k > 1$, an irreducible polynomial of degree k over $\mathbb{Z}_p$).

Proof that Thm. 2 $\Rightarrow$ Thm. 3: Apply Thm. 3 with

$$q(x_{1,1}, \ldots, x_{k,k}) = \text{Perm} \begin{pmatrix} x_{1,1} & \cdots & x_{1,k} \\ . & \cdots & . \\ . & \cdots & . \\ x_{k,1} & \cdots & x_{k,k} \end{pmatrix} \bmod p,$$

$d = \deg(q) = k$, $\mathbb{F} = \mathbb{Z}_p$, $f = A$. The key point is that the condition that A computes ModPerm with probability $> 1 - 1/3(k+1)$ is the same as saying that f agrees with q in a $> 1 - 1/3(d+1)$ fraction of points. Thus, by Theorem 3, given oracle access to A, we can compute $q =$ ModPerm everywhere w.h.p. in time poly$(k, \log p) =$ poly(k). Replacing the oracle calls with the poly-time algorithm for A, we get the desired poly-time algorithm A'. ■

Proof sketch of Thm. 3: Idea: Random Self-Reducibility.
Reduce computing q at an arbitrary point to computing q at random points, where $q = f$ w.h.p. so we can use the oracle f. Finite field $\mathcal{F}^m$ space:



Given x, l = random line through x. Every point on l (other than x) is random, uniform in $\mathcal{F}^m$. $q|_l$ becomes univariate polynomial of deg $\leq d$. If we know the values of q at $d+1$ points on the line, can reconstruct formula to get value at x using polynomial interpolation. ■