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

Lecture Notes 16

March 29, 2010 **Scribe: Önder Eker**

1 Agenda

Average-case complexity:

- recap
- **#P**
- poly reconstruction
- **PSPACE**, **EXP**
- low-degree extensions

2 Recap

We have seen the average case complexity of computable functions with respect to some distribution which is indexed by input length.

Definition 1 *$f : \{0,1\}^* \to \{0,1\}^*$ is* $\delta(n)$-hard *under distribution D_n (on $\{0,1\}^n$) if for every PPT A*

$$\Pr_{\substack{x \leftarrow D_n \\ coins(A)}} [A(x) \neq f(x)] > \delta(n) \quad \textit{for sufficiently large } n$$

f is $\delta(n)$-easy *if*

$$\Pr_{\substack{x \leftarrow D_n \\ coins(A)}} [A(x) = f(x)] \geq \delta(n) \quad \textit{for sufficiently large } n$$

NB:

1. These two definitions are not exactly complementary. The difference arises due to quantification over n. Negation of hardness gives easiness on infinitely many inputs.

$$\begin{aligned} \neg\delta(n)\text{-hard} \quad &\not\equiv \quad (1-\delta(n))\text{-easy} \\ \neg\delta(n)\text{-hard} \quad &\Longleftrightarrow \quad (1-\delta(n))\text{-easy for infinitely many n} \end{aligned}$$

2. For deterministic algorithms, 1-easy is the same as being easy in the worst case (i.e. $f \in$ **FP**). For a randomized algorithms, worst-case easiness (**BPFP**) means being solvable with high probability, but not probability 1, on all inputs.

$$\text{1-easy for deterministic algorithm} \iff \text{in } \mathbf{FP}$$

3 Open/Difficult for NP

We would like to relate worst-case and average-case complexities of **NP**. This is an important, at the same time open and seemingly difficult problem. For example, it would be a major breakthrough to show:

$\mathbf{NP} \not\subseteq \mathbf{BPP} \implies \mathbf{NP}$ is not $(1-1/\text{poly}(n))$-easy under some poly-time samplable distribution D_n

The RHS is a prerequisite for cryptography. It would be great if we could base cryptography on the worst-case hardness of **NP** (e.g. $\mathbf{NP} \not\subseteq \mathbf{BPP}$), since we have a lot of confidence about that. But since we haven't been able to prove implications like the above, cryptography is instead based on stronger assumptions, such as the existence of "one-way functions."

4 For #P

Although relating worst-case and average complexities for **NP** is an open problem, it can be done for some higher classes. Today we will do it for #**P**, **PSPACE**, **EXP**.

The class **BPFP** is akin to **BPP**, but for functions instead of languages.

Theorem 2 *If* $\#\mathbf{P} \not\subseteq \mathbf{BPFP}$*, then there is a function* $f \in \mathbf{FP}^{\#\mathbf{P}}$ *and a polynomial p(n) such that f is not* $(1-1/\text{poly}(n))$*-easy under* U_n*=uniform distribution on* $\{0,1\}^n$

It is possible improve the $1/\text{poly}(n)$ significantly. See the hardness amplification chapter in Arora–Barak.

By a similar proof, it can also be shown that $\#\mathbf{P} \not\subseteq \underbrace{\text{i.o.-}\mathbf{BPFP}} \implies f$ is $1/\text{poly}(n)$-hard, where **i.o.-BPFP** is the class of functions that can be computed in probabilistic polynomial time with bounded error for infinitely many input lengths.

Key ideas:

1. $\text{Perm}(X)$ is a low-degree polynomial in n^2 entries of X. The degree is at most n.
2. Can efficiently reconstruct low-degree polynomials from noisy versions. If there is an algorithm that computes the permanent in most places, then it is possible to compute it everywhere.

5 Polynomial Reconstruction

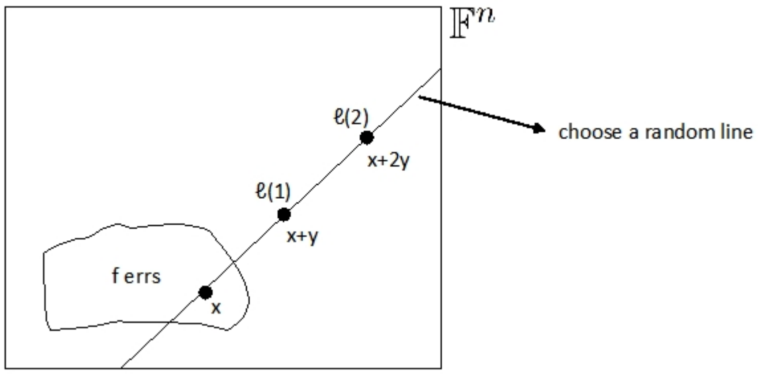
Polynomial reconstruction is an important topic with applications not restricted to average-case complexity.

Theorem 3 *Let* $\mathbb{F}$ *be a finite field (e.g.,* $\mathbb{Z}_p = \{0,\ldots,p-1\}$ *with arithmetic modulo p for a prime p), f:* $\mathbb{F}^n \to \mathbb{F}$ *is a function that agrees with some polynomial* $q : \mathbb{F}^n \to \mathbb{F}$ *of degree at most d in at least* $1-1/(3(d+1))$ *fraction of inputs. Then we can compute q on any input x with high probability in time* $\text{poly}(\log|\mathbb{F}|, n, d)$ *with oracle access to f. That is,* $\exists$ *PPT A such that* $\forall |\mathbb{F}| \geq d+2$ *and n, d, f as above,* $\forall x \in \mathbb{F}^n$

$$\Pr_{coins(A)}\left[A^f(\mathbb{F}, 1^n, 1^d, x) = q(x)\right] \geq \frac{2}{3}$$

Again, a field is set with addition and multiplication satisfying the usual properties, and also with the property that all nonzero elements have multiplicative inverses (i.e. division by nonzero elements is possible). This is why we need p to be a prime. We are giving as input the description of $\mathbb{F}$ that enables doing computations in $\mathbb{F}$ efficiently. This is just the prime p for $\mathbb{Z}_p$. See the Arora–Barak appendix on finite fields for more information about this.

Proof:



Oracle f makes errors in some arbitrary region. We don't know whether a given input x falls into this region. Idea is *random self-reducibility*: we reduce computing q on arbitrary inputs to computing q on random inputs. Specifically, $A^f(x)$ works as follows:

1. Choose a random line $l : \mathbb{F} \to F^n$ through x

$$l(t) = x + ty \qquad \text{for } y \overset{\text{R}}{\leftarrow} \mathbb{F}^n$$

   where $l(0)$ is the point x. Note that $q(l(t))$ is a univariate polynomial of degree d, and for every $i \in \mathbb{F} \setminus \{0\}$, $q(\ell(i))$ is a uniformly random element of $\mathbb{F}^n$.

2. Query the oracle. With probability $\geq 2/3$ we have

$$f(l(1)) = q(l(1)) \quad \wedge \quad f(l(2)) = q(l(2)) \quad \wedge \quad \ldots \quad \wedge \quad f(l(d+1)) = q(l(d+1))$$

3. Interpolate to get $q(l(0))$. We have values of a degree d polynomial on $d+1$ points. We use Lagrange interpolation.

■

5.1 Lagrange Interpolation Formula

$\mathbb{F}$ a field, $\alpha_1, \ldots, \alpha_{d+1} \in \mathbb{F}$ distinct, $\beta_1, \ldots, \beta_{d+1} \in \mathbb{F}$. Then there is a unique polynomial of deg $\leq d$ such that $r(\alpha_i) = \beta_i$ for all i, namely:

$$r(t) = \sum_{i+1}^{d+1} \delta_i(t)\beta_i,$$

where $\delta_i(t)$ is the degree d polynomial:

$$\delta_i(t) = \prod_{j \neq i} \frac{t - \alpha_j}{\alpha_i - \alpha_j} = \begin{cases} 1 & \text{if } t = \alpha_i \\ 0 & \text{if } t = \alpha_j \text{ for } j \neq i. \end{cases}$$

Proof of Theorem 2 using proof 3: (Sketch)
We need a finite field, so we will use the modular version of the permanent. f will be

$$\text{ModPerm}(1^k, 1^p, M) = \text{perm}(M) \bmod p$$

where $p \in \{k+2, \ldots, k^2\}$ prime and $M \in \mathbb{Z}_p^{k \times k}$. Then the distribution D_k is parametrized by k as follows:

1. pick a random prime $p \leftarrow \{k+2, \ldots, k^2\}$
2. pick (uniformly at random) $M \leftarrow \mathbb{Z}_p^{k \times k}$

We can ensure support of $D_k \subseteq \{0,1\}^{n(k)}$ *i.e.*, they can be packed into a string of length n by padding and encoding instances.

Suppose ModPerm is $\left(1 - \frac{1}{3(k+1)^3}\right)$-easy on D_k. Then we have PPT A such that

$$\forall k \quad \Pr_{p, M, coins(A)} [A(1^k, 1^p, M) \neq \text{perm}(M) \pmod p] \leq \frac{1}{3(k+1)^3},$$

where p and M are generated according to D_k.

Observe that there are at most k^2 choices of prime p and the error probability is much smaller than $1/k^2$. Thus, the algorithm must do well for every prime. Specifically, we have:

$$\forall k \forall p \quad \Pr_{M, coins(A)} \left[A(1^k, 1^p, M) \neq \text{perm}(M) \pmod p \right] \leq \frac{1}{3(k+1)}$$

Now we can apply Theorem 3 with $d = k$ to get an algorithm that works for *every* M with high probability. (Note that the permanent of a $k \times k$ matrix is a polynomial of degree k.) We have a PPT A' such that

$$\forall k \forall p \forall M \in \mathbb{Z}_p^{k \times k} \quad \Pr_{coins(A')} \left[A'(1^k, 1^p, M) = \text{perm}(M) \bmod p \right] \geq \frac{2}{3}$$

In the last step we compute the non-modular permanent by using the Chinese remainder theorem. It is #**P**-complete for 0/1 matrices. We compute the permanent for each prime and do error reductions. We have PPT A'' such that

$$\forall k \forall M \in \{0,1\}^{k \times k} \quad \Pr_{coins(A'')} \left[A''(1^k, M) = \text{perm}(M) \bmod \underbrace{p_1 \times \ldots \times p_t}_{\text{primes in } [k+2, \ldots, k^4]} \right] \geq 1 - 2^{-k}.$$

The product of primes is much larger than $k!$ due to their density. (By the Prime Number Theorem, there are $\Omega(k^2/\log k)$ primes in the interval $\{k+2, \ldots, k^2\}$, so their product is $k^{\Omega(k^2/\log k)} \gg k!$.) Therefore the modular reduction does nothing.

$$\Pr_{coins(A'')} \left[A''(1^k, M) = \text{perm}(M) \right].$$

■

5.2 Interpretations of Theorems 2 & 3

- worst-case/average-case equivalence for Perm (or other low degree polynomials)
- "program corrector" for Perm (or other low-degree polynomials, such as Det). We can write a program that gives right answer on most input and correct it so that it gives right answers everywhere. For example, we can write a complicated but fast algorithm for the determinant that may have some bugs in corner cases and then correct. We remark that there is a "complementary" notion of *program testing*, which tests whether a program for a given function is correct on most inputs; thus we can also test whether our program is correct often enough to apply the program corrector.
- polylogarithmic-time local decoding algorithm for Reed-Muller error-correcting code (polynomials of degree $\leq d$ over $\mathbb{F}$). Encoding data as low-degree polynomials allows reconstructing it if some fraction is corrupted by looking at just polylogarithmically many bits of the data. This is a useful idea that has found a variety of other applications.

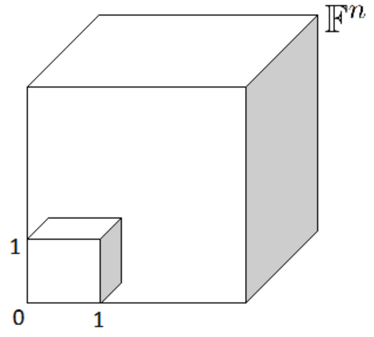
6 Generalizations to other classes: PSPACE, EXP

All we used about the PERMANENT in the previous section is that it is a low-degree polynomial. We now show that there are **PSPACE**-complete and **EXP**-complete low-degree polynomials.

Lemma 4 (multilinear extension) *For every boolean function $f : \{0,1\}^n \to \{0,1\}$ and finite field $\mathbb{F}$, there is a (unique) polynomial $q : \mathbb{F}^n \to \mathbb{F}$ such that*

1. *$q|_{\{0,1\}^n} = f$.*
2. *q has degree ≤ 1 at each variable (and hence has total degree $\leq n$.)*
3. *$q(x)$ can be computed in linear space given $\mathbb{F}, 1^n, x \in \mathbb{F}^n$ and oracle to f.*

Note that 0 and 1 are elements of any field. We extend a function on boolean hypercube to a low degree polynomial on $\mathbb{F}^n$. The first and third properties imply that if f is **PSPACE**-complete (resp. **EXP**-complete), then so is q. Thus **PSPACE** and **EXP** have complete problems that are low-degree polynomials, and hence have worst-case/average-case equivalences. We can also get boolean functions with worst-case/average-case equivalences by taking $q'(x, i)$ to be the i'th bit of q.



Proof of lemma: We interpolate a **PSPACE**-complete problem f as follows:

$$q(x) = \sum_{\alpha \in \{0,1\}^n} \delta_\alpha(x) \cdot f(\alpha)$$

, where $\delta_\alpha(x)$ a polynomial of degree at most 1 in each variable such that

$$\delta_\alpha(\beta) = \begin{cases} 1 & \text{when } \beta = \alpha \\ 0 & \text{when } \beta \in \{0,1\}^n \setminus \alpha \end{cases}$$

$\delta_\alpha(x)$ can be described explicitly as illustrated by the following example:

$$\delta_{1001}(x_1, \ldots, x_4) = x_1 \cdot (1 - x_2) \cdot (1 - x_3) \cdot x_4.$$

■

Note that the construction of q from f requires exponential summation. We can't do that in **NP** but it is easy at complexity classes from #**P** and above.