6.841: Advanced Complexity Theory Fall 2012

Lecture 14 — October 30, 2012

Prof. Dana Moshkovitz *Scribe: Jon Schneider*

1 Overview

In the previous lecture, we began our discussion of pseudorandomness. We presented the Blum-Micali definition of a pseudorandom generator, which defines pseudorandomness in terms of how hard it is for members of a specific computation class to distinguish between true randomness and generated randomness. We proved Yao's theorem, which provides a link between the existence of certain pseudorandom generators and the equivalence of certain complexity classes. Finally, we presented the Nisan-Wigderson pseudorandom generator which (if the Nisan-Wigderson assumption holds) will imply that $BPP = P$.

In this lecture we will complete the proof that the Nisan-Wigderson assumption implies that $BPP = P$. We will then discuss the relation between worst case hardness and average case hardness, and present a hardness amplification result due to Sudan, Trevisan, and Vadhan based on error correcting codes.

2 Nisan-Wigderson Generator

We begin by reviewing the notation of the previous lecture. Recall that a function $f : \{0, 1^k \to \{0, 1\}$ is ϵ-easy for circuits of size m if there exists a circuit C of size at most m such that

$$\Pr_{x \leftarrow \{0,1\}^k}[f(x) = C(x)] \geq \frac{1}{2} + \epsilon$$

The Nisan-Wigderson assumption is that there exists an $f : \{0, 1\}^k \to \{0, 1\}$ such that f is computable in time $2^{O(k)}$ but such that f is not ϵ-easy for circuits of size $2^{2\delta k}$ (for some constant δ).

If such a function exists, we can construct the following pseudorandom generator. Fix $s = \Theta(\log n)$ and n subsets $T_1, T_2, \ldots, T_n \subset \{1, 2, \ldots, s\}$ such that $|T_i| = k$ for each i. Our generator G is then the function from $\{0, 1\}^s$ to $\{0, 1\}^n$ defined by

$$G(z)_i = f(z|_{T_i})$$

where $z|T_i$ is the k-bit string formed by restricting z to the indices in T_i.

In our proof last lecture, we defined the $n + 1$ hybrid distributions H_i for $0 \leq i \leq n$, where H_i is the distribution resulting from concatenating the first i digits of $G(z)$ (for uniformly chosen z) with a uniformly chosen string of length $n - i$. Then, if G is not a pseudorandom generator, there is a distinguisher D of size less than size $2^{2\delta k}$ such that

$$\Pr_{z \leftarrow \{0,1\}^s}[D(G(z)) = 1] - \Pr^n_{y \in \{0,1}[D(y) = 1] > \epsilon$$

If we then let $\alpha_i = \Pr_{w \leftarrow H_i}[D(w) = 1]$, then this condition is simply that $\alpha_n - \alpha_0 > \epsilon$. By the pigeonhole principle, there must exist an i such that $\alpha_i - \alpha_{i-1} > \epsilon/n$. We will show that this results in a contradiction; specifically, we will construct a small circuit that approximately solves f, which contradicts our hardness assumption.

In the previous lecture, we showed how to prove this for $i = 1$. In this lecture, we will show how to extend this to arbitrary i. Let y be a random n bit string; then we can write the claim $\alpha_i - \alpha_{i-1} > \epsilon/n$ as:

$$\Pr_{z,y}[D(f(z|_{T_1}), \ldots, f(z|_{T_i}), y_{i+1}, \ldots, y_n) = 1] - \Pr_{z,y}[D(f(z|_{T_1}), \ldots, y_i, y_{i+1}, \ldots, y_n) = 1] > \frac{\epsilon}{n}$$

Since pseudorandomness is equivalent to unpredictability (a simple corollary of Yao's theorem), the above claim holds iff there exists a circuit D' (about as large as D) such that

$$\Pr_z[D'(f(z|_{T_1}), f(z|_{T_2}), \ldots, f(z|_{T_{i-1}})) = f(z|_{T_i})] \geq \frac{1}{2} + \frac{\epsilon}{n}$$

Note that the right hand side of $D'(f(z|_{T_1}), f(z|_{T_2}), \ldots, f(z|_{T_{i-1}})) = f(z|_{T_i})$ only depends on $z|_{T_i}$. Therefore, letting $\bar{T_i}$ be the complement of T_i in $\{1, \ldots, s\}$, by the averaging principle, there must be a setting for $z|\bar{T_i}$ such that the claim still holds. Fixing $z|_{\bar{T_i}}$ to this setting, we can write $f_j(z|_{T_i \cap T_j}) = f(z|_{T_j})$ for $1 \leq j < i$. Our claim then becomes:

$$\Pr_{z|T_i}[D'(f_1(z|_{T_1 \cap T_i}), f_2(z|_{T_2 \cap T_i}), \ldots, f_{i-1}(z|_{T_{i-1} \cap T_i})) = f(z|_{T_i})] \geq \frac{1}{2} + \frac{\epsilon}{n}$$

Now, we wish to compute $f(z')$ easily. To achieve this, we will choose our sets T_i so that all the intersections $T_i \cap T_j$ are small. Then, since any function on ℓ variables can be simulated by a circuit of size $\ell 2^\ell$, if these intersections are small then we should be able to easily simulate all of the functions f_j.

In particular, if $|T_i \cap T_j| \leq \delta k$, then there exists a circuit D'' of total size $i\delta k 2^{\delta k} + |D'| \leq 2^{2\delta k}$ which satisfies

$$\Pr_{z|_{T_i}}[D''(z|_{T_i}) = f(z|_{T_i})] \geq \frac{1}{2} + \frac{\epsilon}{n}$$

But this contradicts our hardness assumption for f! Therefore, as long as we can show that such sets T_i exist, then our main result follows. In particular, we must show the following lemma.

Lemma 1. *For every positive k and δ, there exists an $s = O(k/\delta)$ and $n = 2^{\Omega(\delta^2 k)}$ and sets $T_1, ..., T_n \subset \{1, \ldots, s\}$ such that $|T_i| = k$ and $|T_i \cap T_j| \leq \delta k$. Moreover, we can find such a set in polynomial time.*

Such collections of subsets are known as *combinatorial designs*. There exists a simple greedy algorithm for finding such sets; for more details, interested readers should consult Section 20.2 of the class textbook.

3 Worst case versus average case hardness

Most of the time in theoretical computer science, we are concerned with worst case hardness; for example, when we talk about algorithms, we often measure their efficiency by their worst-case running times, and hard problems like SAT need only be hard in the worst case.

In our discussion of pseudorandomness above, we needed problems that are hard not just in the worst-case, but in the average-case. For example, our definition of a function being ϵ-easy to compute was in terms of the probability over random k-bit strings chosen uniformly at random. In order to prove the validity of the Nisan-Wigderson assumption, we will therefore need problems that are not just hard in the worst case, but hard in the average case.

3.1 The Permanent

Unfortunately, it turns out that it is usually much harder to prove claims about average case complexity than it is to prove claims about worst case complexity. One problem that we can prove average case complexity claims about is the problem of computing the *permanent* of a matrix.

Definition 2. *If M is an n by n matrix with entries M_{ij}, the permanent of M is given by*

$$\text{perm}(M) = \sum_{\pi} \prod_{i=1}^{n} M_{i\pi(i)}$$

where the sum is over all $n!$ permutations of the numbers 1 through n.

Note that the above formula for the permanent is very similar to the Leibniz formula for the determinant, the exception being that each product has an additional $(-1)^{\text{sgn}(\pi)}$ factor. However, while the determinant can be computed in polynomial time (via Gaussian elimination, for example), the permanent is very hard to compute. It is known (due to a theorem of Valiant) that computing the permanent is $\#P$ complete [4].

It turns out (due to an argument of Lipton) that computing the permanent is also hard on average. In particular, we have the following theorem.

Theorem 3. *If algorithm A computes $\text{perm}(M) \bmod p$ in expected time $t(n)$ when M is uniformly chosen at random from $\mathbb{Z}_p^{n \times n}$, then there exists a probabilistic algorithm that computes the permanent $\text{perm}(M)$ on all M in time $\Theta(nt(n))$*

Proof. See [4]. □

Based on this observation, we might wonder whether we can link average-case hardness to worst-case hardness in any other situations.

3.2 Hardness Amplification and Error Correcting Codes

Recall that **E** is the complexity class given by $DTIME(2^{O(n)})$. The following theorem is due to Sudan, Trevisan, and Vadhan.

Theorem 4. *([3]) If there exists an $f : \{0,1\}^* \to \{0,1\}$ such that $f \in \mathbf{E}$ and f is not computable in size $2^{\delta n}$, then there exists a function $\tilde{f}$ such that $\tilde{f} \in \mathbf{E}$ but $\tilde{f}$ is not ϵ-easy to compute in size $poly(\epsilon)2^{\delta n}$.*

Note that this essentially says that if we have a function that is hard to compute in the worst-case (not computable in size $2^{\delta n}$) then it is approximately as hard to compute in the average case (not ϵ-computable in size $poly(\epsilon)2^{\delta n}$).

The proof of this theorem is based on the theory of error-correcting codes.

Definition 5. *A τ-error-correcting code is a pair of functions*

$$E : \{0,1\}^k \to \{0,1\}^n$$

and

$$D : \{0,1\}^n \to \{0,1\}^k$$

that satisfy the relation

$$D(E(x) \oplus \eta) = x \ \forall \eta \in \{0,1^n \ s.t. \mathrm{wt}(\eta) \leq \tau n$$

.

Here $\oplus$ is the XOR operation, and $\mathrm{wt}(s)$ for a binary string s is equal to the number of bits in s set to 1.

As suggested by the name, error-correcting codes are primarily used to store information with some redundancy so the information is resilient to some amount of noise. For example, for a τ-error-correcting code, if you toggle at most a fraction of τ of the bits after you encrypt a piece of information, then you can still extract the original message.

Clearly n must be larger than k. A simple information theoretical argument shows that, for a τ-error-correcting code n must be at least $k/(1 - H(\tau))$, where $H(\tau) = -\tau \log \tau - (1-\tau)\log(1-\tau)$ is the one-bit information entropy of τ (the Gilbert-Varshamov Bound further shows that such codes exist for all $\tau < 1/2$). Since $H(1/2) = 1$, this bound also shows that codes can only exist for $\tau < 1/2$. It is easy to see that in general, codes cannot exist for $\tau > 1/2$; in particular, since two n-bit words w_1 and w_2 can disagree in at most n bits, by toggling the first half of these bits we have a word w' that is within Hamming distance $n/2$ of both w_1 and w_2.

For our purposes, we also want E and D to be computable in polynomial time. For all $\tau < 1/4$, there are known explicit constructions of polynomial-time-computable τ-error correcting codes with $n = O(k)$.

It turns out that it is very hard to beat the $\tau = 1/4$ barrier. We will therefore consider the following relaxed definition of an error correcting code (known as a *list decodable code*).

Definition 6. *A (τ, L)-error-correcting code is a pair of functions $E : \{0,1\}^k \to \{0,1\}^n$ and $D : \{0,1\}^n \to \{\{0,1\}^k\}^L$ such that*

$$x \in D(E(x) \oplus \eta)\ \forall \eta \in \{0,1\}^n \ s.t. \mathrm{wt}(\eta) \leq \tau n$$

.

In particular, our local decoder D now generates L possible messages, and we only require that the actual message belongs to this set. For these error-correcting codes, the following theorem is known.

Theorem 7. *For every $\tau \leq 1/2 - \epsilon$, there exists a polynomial-time computable $(\tau, L = O(1/\epsilon^2))$ error-correcting-code with $n = O(k/\epsilon^3)$.*

The rough idea is that to transform a worst-case hard function f to an average-case hard function $\tilde{f}$, we treat the truth table of f as a message that is to be encoded using an error correcting code. The encoded version then is the truth table of an average-case hard function $\tilde{f}$. To prove average-case hardness, one shows that any efficient algorithm A that solves $\tilde{f}$ on a $1/2 + \epsilon$ fraction of inputs can be treated as a "corrupted" version of $\tilde{f}$, and then combined with a decoding procedure, f can be computed exactly using the efficient algorithm – which would contradict worst-case hardness of f.

However, even list decoding is not enough for this process to work; we require that the decoding procedure runs extremely efficiently (so that A combined with the decoding procedure is still efficient). Thus we need a stronger notion called *local list decoding*, where the decoding procedure runs in *sublinear time* (it doesn't need to read the the entire codeword in order to decode the message). We refer the readers to [3] for more details.

We can now provide a proof sketch of Theorem 4.

Proof of Theorem 4. Let (E, D) be a $1/2 - \epsilon$ local list-decoding code (for some ϵ to be determined later) whose decoder runs in sublinear time, where $E : \{0,1\}^K \to \{0,1\}^N$, with $K = 2^k$. Then a function $f : \{0,1\}^k \to \{0,1\}$ can be viewed as a K-bit string and thus and element of the domain of E. $E(f)$ is then an N-bit string, which can be interpreted as the truth table of a function $\tilde{f} : \{0,1\}^n \to \{0,1\}$. Since $\tilde{f}$ is some function on n binary inputs, it can be computed in time $2^{O(n)}$ and therefore $\tilde{f} \in E$.

We need to show that this $\tilde{f}$ is hard to ϵ-approximate. Note that if you can compute an ϵ approximation to $\tilde{f}$, then the decoder D will be able recover f exactly. This means that if f is hard, then $\tilde{f}$ should be hard to approximate, which is exactly the statement that we want.

□

References

[1] R. Lipton, *New directions in testing*, Distributed Computing and Cryptography, 2:191-202, 1991.

[2] N. Nisan, A. Wigderson, *Hardness vs randomness*, Journal of Computer and System Sciences, 49(2):149-167, 1994.

[3] M. Sudan, L. Trevisan, and S. Vadhan, *Pseudorandom generators without the XOR lemma* J. Comput. Syst. Sci., 62(2):236-266, 2001.

[4] L. G. Valiant, *The Complexity of Computing the Permanent*, Theoretical Computer Science 8(2):189201, 1979.