6.841 Advanced Complexity Theory	May 02, 2005
Lecture 23	
Lecturer: Madhu Sudan	*Scribe: Elena Grigorescu*

1 Introduction

In today's lecture we will present the Nisan& Wigderson's pseudorandom generator, which is based on the assumption of the existence of a 'hard on the average' function. The construction can be used to derandomize computation and it would imply $BPP = P$. If we consider the 'hard on the average' requirement as a strong assumption, it turns out that weaker assumptions can be imposed. In particular, Impagliazzo-Wigderson ('97) and Sudan-Trevisan-Vadan ('99) show that a 'hard on the worst case' function implies a 'hard on the average' function. We present a proof of the first theorem, while we only state formally the latter.

Recall from the last lecture that

Definition 1 *$G : \{0,1\}^l \to \{0,1\}^n$ is an (ϵ, s)-pseudorandom generator if $\forall$ circuits C of size $\leq s$ we have that*

$$|Pr_{x \leftarrow U_n}[C(x) = 1] - Pr_{\sigma \leftarrow U_l}[C(G_1(\sigma)) = 1]| \leq \epsilon.$$

2 Nisan-Wigderson

We start by defining what it means to be hard on the average. In this lecture we only deal with hardness over the uniform distribution, while in future lectures we might extend the notion to other distributions as well.

Definition 2 *(Hardness on the average)*
Let the family f be $f_n : \{0,1\}^n \to \{0,1\}$. We say that f_n is $(\alpha, s) -$ hard if for every circuit C of size $\leq s$ we have that $Pr_{x \leftarrow U_n}[f(x) \neq C(x)] \geq \alpha$.

Notice that in the worst case $\alpha = 2^{-n}$. Also, since every boolean function can be computed with probability 1/2 on random instances, it follows that $\alpha < 1/2$. Therefore, the larger α and s the harder the function.

We can now state the main theorem.

Theorem 3 (Nisan-Wigderson) *If $\exists$ a function $f : \{0,1\} \to \{0,1\}$ that is $(1/2 - 2^{-\delta t}, 2^{\delta t})$-hard but f is computable in $E = \cup DTIME(2^{ct})$ then $\exists$ $G : \{0,1\}^{O(\log n)} \to \{0,1\}^n$ that is a $(1/10, n)$-pseudorandom generator.*

Informally, this states that from a hard function which is computable in exponential time, one can build a pseudorandom generator that can stretch a $O(\log n)$ seed to n bits, and whose output cannot be distinguished from random by any circuit on n gates. To decide a BPP language one only needs to enumerate all the seeds of G (this takes poly time) and treat them as the random choices of the BPP. Therefore, it would follow that $BPP = P$.

As mentioned in the Introduction, [IW97] were able to show the following.

Theorem 4 (Impagliazzo-Wigderson) *(Hardness Amplification theorem)*
If there exists $f : \{0,1\}^t \to \{0,1\}$, $f \in E$ and f is not computable by size $s = 2^{\epsilon t}$ circuits, then $\exists \hat{f} : \{0,1\}^{\hat{t}} \to \{0,1\}$ that is computable in E but st $\hat{f}$ is $(1/2 - 2^{\delta \hat{t}}, 2^{\delta \hat{t}})$-hard.

We will not prove this last theorem in class.

2.1 Short Digression

One setting where we previously used an approach similar to the IW theorem was when we presented Lipton's result regarding the computation of the permanent of a matrix. Recall that if there exists a poly time algorithm that computes $perm(A)$ on a random instance then there exists a randomized algorithm that computes the permanent. The implication was of the form $perm(A) \to f(\theta) = perm(A + \theta R)$, for $\theta \neq 0$. Lipton's reduction states that $perm(A)$ is $(1/poly(t), exp(t))$-hard. If for some random R we get wrong values of the permanent, we can correct these errors using standard techniques in coding theory. Lipton's result and 'standard Reed Solomon Decoding techniques' give us that $perm$ is $(1/2 - 1/poly(t), exp(t))$-hard. However, since $perm$ is not a boolean function 1/2 is not a natural threshold. Using 'List Decoding of the Reed Solomon codes', we may conclude that in fact $perm$ is $(1 - 1/exp(t), exp(t))-$hard.

2.2 Proof of the NW theorem

Given $f : \{0,1\}^t \leftarrow \{0,1\}$ we aim to use it repeatedly n times in order to generate n-bit strings. Therefore, we need a way to generate $\sigma_1, \sigma_2, \ldots, \sigma_n$ using σ, such that $G(\sigma) = f(\sigma_1)f(\sigma_2)\ldots f(\sigma_n)$.

One attempt that does not work is the $Blum - Micali$ approach, which we recall in here. Given f, σ, compute $f(\sigma)$. Apply f to the last $|\sigma|$ bits of $\sigma f(\sigma)$ and repeat the process, always applying f to the last $|\sigma|$ bits of the new concatenated output. Thus we expand the seed bit by bit. Let $g(\sigma) = (\sigma, f(\sigma))$. BM show that this algorithm works well if g is a one-way function. However, we cannot use this for our purposes since we cannot show that f is one-way.

NW suggest computing the σ_i's in parallel, rather than sequentially choosing them. Let $|\sigma| = l$. Pick sets of indices $S_1, S_2, \ldots, S_n \subseteq \{1, 2, \ldots, l\}$ st $|S_i| = t$, $t = l/100$, and let $\sigma_i = \sigma$ projected to S_i. The main requirement to impose on the S_i's is that they should not overlap much, otherwise we cannot expect $f(\sigma_i)$ to be pseudorandom. We use $|S_i \cap S_j| \leq t/99$. This is the expected intersection size when we pick the sets S_i at random. In the language of error correcting codes, the sets S_i satisfying the above properties are called 'designs'. It can be shown that computing the S_i's takes poly time. One can simply pick them by brute force one after the other while preserving their properties.

We will now show that if $G(\sigma) = f(\sigma_1)f(\sigma_2)\ldots f(\sigma_n)$ is not a pseudorandom generator then f is not hard. The framework of the proof is the following. Suppose $\exists$ circuit C st $size(C) \leq s$ and st $Pr[C(x) = 1] \leq Pr[C(G(\sigma)) = 1] - \epsilon$. Using a series of distribution over strings of length n, we want to show that $\exists\ \hat{C}$ of size s st $Pr[\hat{C}(y) \neq f(y)] \leq 1/2 - \epsilon/2n$.

We will accomplish this using a Hybridization argument.

Let

$$\begin{aligned}
D_0 = &\quad U_n \text{ (the uniform distribution on } n \text{ bits).} \\
&\quad \vdots \\
D_i = &\quad \text{Pick } \sigma \text{ at random and } b_1, b_2, \ldots, b_n \text{ bits at random} \\
&\quad \text{output first } i \text{ bits of } G(\sigma)\&b_{i+1}, \ldots, b_n. \\
&\quad \vdots \\
D_n = &\quad \{G(\sigma)\}_{\sigma \leftarrow U_n}.
\end{aligned}$$

We assumed that the circuit C distinguishes the outputs of G from random strings, thus

$$Pr_{x \leftarrow D_n}[C(x) = 1] - Pr_{x \leftarrow D_0}[C(x) = 1] \geq \epsilon.$$

Define

$$p_j = Pr_{x \leftarrow D_j}[C(x) = 1] - Pr_{x \leftarrow D_{j-1}}[C(x) = 1].$$

Adding over the p_j's, we obtain

$$\sum_{j=1}^{n} p_j = Pr_{x \leftarrow D_n}[C(x) = 1] - Pr_{x \leftarrow D_0}[C(x) = 1] \geq \epsilon.$$

Then it must be the case that $\exists i$ st $p_i \geq \epsilon/n$. Let $y_i = f(\sigma_1)f(\sigma_2)\ldots f(\sigma_i)b_{i+1}\ldots b_n$, and $y_{i-1} = f(\sigma_1)f(\sigma_2)\ldots f(\sigma_{i-1})b_i b_{i+1}\ldots b_n$.

We can rewrite $D_i = \{y_i\}_{\sigma, b_{i+1}, \ldots b_n}$ and $D_{i-1} = \{y_{i-1}\}_{\sigma, b_i, \ldots b_n}$. We will show that if it is possible to distinguish between these two distributions then it is possible to predict the ith bit. With the new notation we have

$$Pr_{\sigma, b_i, \ldots, b_n}[C(y_i) = 1] - Pr_{\sigma, b_{i+1}, \ldots, b_n}[C(y_{i-1}) = 1] \geq \epsilon/n$$

implies

$$\exists b_{i+1}, \ldots, b_n \text{ st } Pr_{\sigma, b_i, \ldots, b_n}[C(y_i) = 1] - Pr_{\sigma, b_{i+1}, \ldots, b_n}[C(y_{i-1}) = 1] \geq \epsilon/n.$$

Define now a new circuit $C'(v_1, \ldots, v_i) = C(v_1, \ldots, v_i, b_{i+1}, \ldots, b_n)$, with the $b_{i+1}, \ldots, b_n$ hard-wired. Then

$$Pr[C'(f(\sigma_1)f(\sigma_2)\ldots f(\sigma_i)) = 1] \geq Pr[C'(f(\sigma_1)f(\sigma_2)\ldots f(\sigma_{i-1})b_i) = 1] + \epsilon/n,$$

and thus C' predicts $f(\sigma_i)$ from the bits $f(\sigma_1)f(\sigma_2)\ldots f(\sigma_{i-1})$.

Now let σ' be the projection of σ on the coordinates not in S_i, so $\sigma = (\sigma_i, \sigma')$. It can be shown that $\exists \sigma'$ st the distinguishing probability does not decrease, and in what follows fix that σ'. Since all the intersections of the designs are small ($|S_j \cap S_k| \leq t/99 = l/9900$) it follows that $f(\sigma_j)$ is only dependent on a small fraction of the unfixed bits of σ. Define $f_j(\sigma_i) = f(\sigma_j)$. It follows that $f_j(\sigma_i)$ can be computed by a $2^{t/99}$-sized circuit. Let a new circuit

$$C''(\sigma_i, b_i) = C'(f_1(\sigma_i)f_2(\sigma_i), \ldots, f_{i-1}(\sigma_i)b_i).$$

We can conclude that

$$Pr_{\sigma_i}[C''(\sigma_i, f(\sigma_i)) = 1] \geq Pr_{\sigma_i,b_i}[C''(\sigma_i, b_i) = 1] + \epsilon/n,$$

and the size of C'' is $|C''| \leq s + n2^{t/99}$ (where s is the size of the initial circuit C.) Note that this size is still too small to compute $f(\sigma)$. To finalize the proof that f is not hard, build a new circuits $\hat{C}$ which computes $f(x)$, and has access to C''.
$\hat{C}$:
on input x
compute $C''(x, 0)$ and $C''(x, 1)$
if they are equal(then cannot predict better than w.p 1/2) output random bit b
else, output b st $C''(x, b) = 1$.
Therefore, we can conclude that $Pr_x[f(x) = \hat{C}(x)] \geq 1/2 + \epsilon/2n$ which contradicts the hardness of f assumption. As an exercise, show that this probability suffices to derandomize BPP.

2.3 Other results

Using the NW construction Impagliazzo&Kabanets prove:

Theorem 5 *If $\exists$ a deterministic poly time algorithm for polynomial identity testing then Permanent $\notin$(Algebraic P/poly) or $NEXP \not\subseteq P/poly$.*

These results are suggesting the fact that proving circuit lower bounds is equivalent to constructing good pseudorandom generators.