CMPT 881: Pseudorandomness Prof. Valentine Kabanets

Lecture 20: *NW* Pseudorandom Generator

November 25, 2004 *Scribe: Ladan A. Mahabadi*

1 Introduction

In this last lecture of the course, we'll discuss the construction and properties of the Nisan-Wigderson pseudorandom generator. This NW generator plays a crucial role in derandomization and differs from the BMY generator ([5], [6]) discussed in the previous lectures in its running time (exponential in the seed length as opposed to polynomial) and the power of the distinguisher (fooling distinguishers with running time of a fixed polynomial time as opposed to all polynomial time computations)

1.1 One Bit Stretch

Last time, we described a pseudorandom generator that stretched its truly random input by one. If f is $(\frac{1}{2}-\epsilon)$-hard function for time $t(n)$, then $G(x) = x \; o \; f(x)$ is a PRG for time $t(n)$ distinguishers. However, composing the generator with itself several times does not yield a longer stretch.
The adversary in the BMY setting is any polynomial time computation, and thus has enough resources to simulated the polynomial time G but this is not the case for the NW PRG running in exponential (in the seed length) time.

1.1.1 Direct product (second attempt at increasing the stretch parameter)

- *Attempt 2.1*
  $G(x_1, ..., x_k) = x_1 \; o \; ... \; o \; x_k \; o \; f(x_1) \; o \; ... \; o \; f(x_k)$ where next-bit unpredictability ([5], [7]) implies the pseudorandomness of the output of G. The only problem is that the stretch is not too big (say $f : \{0, 1\}^n \rightarrow \{0, 1\}$). In other words the stretch of $kn \rightarrow kn + k$ is not enough since the additive stretch k is not large compared to the investment of kn bits.

- *Attempt 2.2*
  To deal with this problem, Nisan and Wigderson derandomized the previous construction by using combinatorial designs. Instead of using true randomness for x_1, ..., x_k, use pseudorandomness. Namely, use design$(S) = x_1 \; ... \; x_l$ where the following hold:

  $|S| << |x_1| + ... + |x_k|$

  $NW(S) = G(\; Design(S) \;)$ is a PRG.

Definition 1 (Packings or Combinatorial Designs) $s_1, \; ... \; , s_m \subseteq [d]$ *is* (l, a)*-design if*

- $\forall i, |s_i| = l$
- $\forall i \neq j, |s_i \cap s_j| \leq a$
- **Representation of designs using matrices**:
  Given some universe $[d]$, create the masking matrix M where each row denotes the characteristic vector s_i, each column i, and M_{ij} will be 1 if j is picked in s_i and 0 otherwise (each row acts as a filter). In this binary matrix, number of 1's in each row is equal to l, and no two rows have too many common 1's $\leq a$ (where for a fixed l, d is small, a is small, and m is large preferably).

By the probabilistic method, one can prove existence of such designs, and by brute force and picking the sets one after the other while preserving the properties of the design, such designs can be constructed.

Lemma 2 *$\forall$ const $\gamma > 0$, $\forall l, m, n \in \mathbb{N}, \exists$ (l, a)-design $S_1, ..., S_m \subseteq [d]$ with the following parameters:*

- $d = O(\frac{l^2}{a})$
- $a = \gamma \log m$

Moreover, this design can be constructed efficiently in poly(m, d)-time.

Construction of the Nisan-Wigderson PRG [1]

Given (l, a)-design $S_1, ..., S_m \subseteq [d]$ and average-case hard function $f : \{0,1\}^l \to \{0,1\}$, define $NW : \{0,1\}^d \to \{0,1\}^{m+d}$ where $NW(x) = x \ o \ f(x|_{s_1}) \ o \ ... \ o \ f(x|_{s_m})$

Remark 3 Usually the seed x is not outputted by the PRG. In the above construction, we have included it for parallelism with PRG constructions of previous lectures.

Theorem 4 (NW[1]) *Suppose $\exists$ function $f : \{0,1\}^l \to \{0,1\}$ computable in Dtime$(2^{O(l)})$ such that:*

f is $(\frac{1}{2} - \frac{1}{t})$-hard for non-uniform time t. Let $S_1, ..., S_m \subseteq [d]$ be an (l, a)-design with $m = t^{\frac{1}{3}}$, $a = \frac{\log t}{3} = \log m$. Then, $NW : d \to d + m$ is $\frac{1}{m}$-PRG for time m.

Remark 5 m depends on $poly(t)$ which depends on the hardness of function f. If f is exponentially-hard (i.e $t = 2^{\Omega(l)}$), then $d = O(\frac{l^2}{a}) = O(\frac{l^2}{\log t}) = O(\frac{l^2}{l}) = O(l)$ and $m = t^{\frac{1}{3}} = 2^{\Omega(l)}$. In other words, $NW : O(l) \to 2^{\Omega(l)}$

Thus, an exponentially-hard function results in $NW : O(\log n) \to n$. Enumerating over all possible seeds of such a pseudorandom generator, using its output as the random bits used in a BPP algorithm, and outputting the majority outcome, results in $BPP = P$.
In general, for $t = 2^{\Omega(l)}$, can get a $PRG : O(l) \to t(l)$ (Omitting the polynomial factor). Basically, hardness of a function gives pseudorandomness:

$t(l)$-*hardness* $\quad\to\quad$ $t(l)$ bits of *pseudorandomness* out of l truly random bits

Furthermore, the converse is also true. Thus, there is a tight trade off between hardness of boolean functions and the existence of a pseudorandom generator with matching parameters.

Proof: Suppose G is not $\frac{1}{m}$-PRG for time m, reach contradiction for hardness of f. In other words, if G is not $\frac{1}{m}$-PRG for time m, then $\exists$ a distinguisher T such that:

$\Pr_{T(G(U_d))=1} [-] \Pr_{T(U_m)=1} [>] \epsilon = \frac{1}{m}$

Define hybrid distributions $H_i = g(U_d)|_{1\ \dots\ i}\ , U_{m-i}$. Note that $H_m \equiv G(U_d)$ and $H_0 \equiv U_m$.

$\Pr_{T(G(U_d))=1} [-] \Pr_{T(U_m)=1} [=] \Sigma_{i=1}^{m}\ (\Pr_{T(H_i)=1} [-] \Pr_{T(H_{i-1})=1} [])$

So, there must $\exists i$ such that

$$\Pr_{T(H_i)=1} [-] \Pr_{T(H_{i-1})=1} [\geq] \frac{\epsilon}{m} = \frac{1}{m^2} \tag{1}$$

Furthermore, $H_i = G(U_d)|_{1\ \dots\ i}\ , U_{m-i}$ and $H_{i-1} = G(U_d)|_{1\ \dots\ i-1}, U_{m-i+1}$ differ in position i. By an averaging argument, all of the $m - i$ rightmost bits in H_i and H_{i-1} can be fixed so that (1) is preserved.
The new distinguisher needs to have $(m - i)$ bits of non-uniformity. Thus, $H_i = G(U_d)|_{1\ \dots\ i}$ and $H_{i-1} = G(U_d)|_{1\ \dots\ i-1}, U$.
By Yao's Theorem (indistinguishability is equivalent to pseudorandomness [6]), we get a predictor P such that for $X \leftarrow U_d$, the following holds:

$\Pr_{P(G(X)|_{1\ \dots\ i-1})=G(X)_i} [\geq] \frac{1}{2} + \frac{1}{m^2}$

From this predictor, an algorithm for inverting f can be induced. Recall that $G(x)_i = f(x|_{S_i})$. Similarly, $G(x)|_{1\ \dots\ i-1} = f(x|_{S_1})\ \dots\ f(x|_{S_i})$. It follows that for $X \leftarrow U_d$,

$\Pr_{P(f(X|_{S_1})\ \dots\ f(X|_{S_{i-1}})=f(X|_{S_i})} [\geq] \frac{1}{2} + \frac{1}{m^2}$

General Idea:
Given $y \in \{0,1\}^l$, place y in the positions indexed by S_i so that $f(x|_{S_i}) = f(y)$ and the rest of x will be chosen randomly. By averaging, fix all bits of x outside S_i so that the prediction probability is preserved. After this fixing, each $f(X|_{S_j})$ for $j < i$ depends on $\leq a$ bits of $X|_{S_j}$. Each such $f(X|_{S_j})$ has look up table of size $\leq 2^a$ which can be given as another non-uniform piece of advice to our predictor. This results in a new predictor P' of size $m2^a + (sizeP) = O(t^{\frac{2}{3}}) \leq t$ such that

$\Pr_{P'(y)=f(y)}[\geq] \frac{1}{2}+\frac{1}{m^2} \geq \frac{1}{2}+\frac{1}{t^{\frac{2}{3}}} \geq \frac{1}{2}+\frac{1}{t}$

■

This demonstrates how average-case hardness can be used to construct the NW PRG which can be used for derandomization of BPP. However, it is also desired to prove the existence of a PRG strong enough for derandomization of BPP from a weaker assumption, a worst-case hardness assumption.

Definition 6 $f:\{0,1\}^n \to \{0,1\}$ *is worst-case hard if* $\forall$ *non-uniform time* $t(n)$*, algorithm* A *and* $\forall$ *large* n*,* $\exists x \in \{0,1\}^n$ *such that* $A(x) \neq f(x)$*.*

Approach:
Convert worst-case hardness into average-case hardness. This will be an efficient conversion in the following sense:

Given a truth table of a worse-case hard function $f:\{0,1\}^n \to \{0,1\}$ (i.e truth table of size 2^n), compute the truth table (with size $poly(2^n)$) of a new function $f:\{0,1\}^{O(n)} \to \{0,1\}$ in time polynomial in the size of the output (i.e $poly(2^n)$) such that f' is average-case hard

Theorem 7 ([3],[2]) *If there is a function* $f:\{0,1\}^n \to \{0,1\}$ *in* $Dtime(2^{O(n)})$ *which is worst-case hard for non-uniform time* $2^{\Omega(n)}$*, then* $\exists G: O(\log n) \to n$ *and* $BPP = P$

We want to get an average-case $(\frac{1}{2}-\frac{1}{2^{\Omega(n)}})$-hard function on $O(n)$ bits which is hard for non-uniform time $2^{\Omega(n)}$. This conversation from worst-case to average-case hardness is accomplished by using error correcting codes (ECC):

- Represent worst-case hard function f using its truth table of size 2^n
- Encode the function using ECC into a function f' of size 2^{cn}
- The decoder has *access* to f' 's truth table

We want to argue that f' is average-case hard. Suppose $\exists$ algorithm A such that $A(x) = f'(x)$ in lots of positions (i.e A and f' have lots of bits in common). One can error correct the few errors present to get a perfect code word out. Note that the truth table of A will be of size *poly*(size of input to the decoder) = $poly(2^n)$. However, the decoding algorithm does not need to be given the entire message and oracle access suffices. In other words, we need to use *locally decodable* error codes so that value of f' can be output in any specified position in polynomial time. Fortunately, explicit locally-decodable codes are known.
Thus, the number of errors in this case is $\frac{1}{2}-\frac{1}{2^{\Omega(n)}}$. In other words, there are $\frac{1}{2}+\frac{1}{2^{\Omega(n)}}$ agreements between the true value of f' and the output of algorithm A, but this amount of agreement can not guarantee a unique value for f' [4]. Hence, we need to use *list-decodability*: instead of a single codeword, we output a "small" list of codewords that are close to f'.

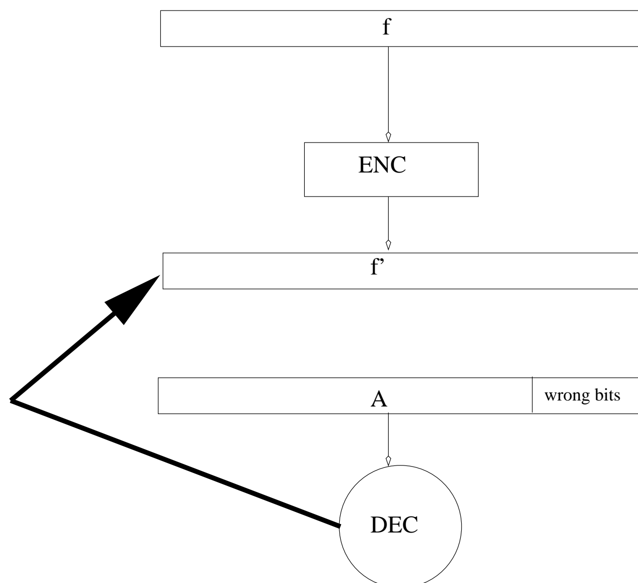

Figure 1: Worst-case to Average-case Conversion through Error Correcting Codes

The conversion of worst-case hardness to average-case hardness was only briefly sketched. The actual construction and proofs are very elegant and have found other applications in complexity theory.

References

[1] N. Nisan and A. Wigderson, **Hardness vs randomness**, Journal of Computing and Systems Sciences, 49(2):149–167, Oct. 1994.

[2] R. Impagliazzo, A. Wigderson,**P=BPP unless E has Subexponential Circuits: Derandomizing the XOR Lemma**, Proc. of the 29th STOC, pp. 220-229, 1997.

[3] M. Sudan, L. Trevisan, and S. Vadhan, **Pseudorandom generators without the XOR lemma**, In Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pages 537-546, 1999.

[4] Madhu Sudan, **List decoding of polynomial codes**, (Covers joint works with Sanjeev Arora, Venkatesan Guruswami, Luca Trevisan and Salil Vadhan.)

[5] M. Blum and S. Micali,**How to Generate Cryptographically Strong Sequences of Pseudo-random Bits** SIAM Journal on Computing, 13, no 4, 1986, 850–864.

[6] A.C. Yao,**Theory and applications of trapdoor functions**, In Proceedings of the Twenty-Third Annual IEEE Symposium on Foundations of Computer Science, pages 80-91, 1982.

[7] Oded Goldreich, **Foundations of Cryptography (Fragments of a Book)**