Lecture 27

In this lecture we will look at the Nisan-Wigderson construction of pseudorandom generators using a language L that has a $2^{O(n)}$ time algorithm but no circuit family of size less than $2^{\epsilon n}$ can identify strings in L with a significant advantage. The construction uses subset families with some desired properties, and the proof uses techniques used in the Blum-Micali construction discussed in lecture 13.

1 Notation

If L is a language, then for any n, L_n is the restriction of L to inputs of length n, viewed as a boolean function :

$$L_n : \{0,1\}^n \to \{0,1\}$$

$$L_n(x) = 1 \text{iff} x \in L$$

For a function $f : \{0,1\}^n \to \{0,1\}$, the *hardness* of f, $H(f)$ is the largest value of S such that for every circuit C of size at most S,

$$\mathbf{Pr}_x[C(x) = f(x)] \leq \frac{1}{2} + \frac{1}{S}$$

2 The main result

Theorem 1
[Nisan Wigderson 1988] *Suppose there is $L \in DTIME(2^{O(n)})$ and an $\epsilon > 0$ such that for all sufficiently large n, $H(L_n) \geq 2^{\epsilon n}$. Then there is a family of pseudorandom generator $G_n : \{0,1\}^{t(m)} \to \{0,1\}^m$ computable in time polynomial in m with $t(m) = O(\log m)$. Further, then, $\mathbf{P} = \mathbf{BPP}$.*

The construction of the generator will be based on inputs of L of logarithmic length. We will choose the length of the inputs so that computing L_n can be done in polytime, but the smallest circuit that can compute L_n with a non-negligible advantage is of size $\omega(n^2)$. Set $l = \frac{2\log m}{\epsilon}$. Then

- L on inputs of length l can be solved in time $2^{O(l)} = m^{O(\frac{1}{\epsilon})}$.
- $L_l : \{0,1\}^l \to \{0,1\}$ has hardness $2^{\epsilon l} = m^2$.

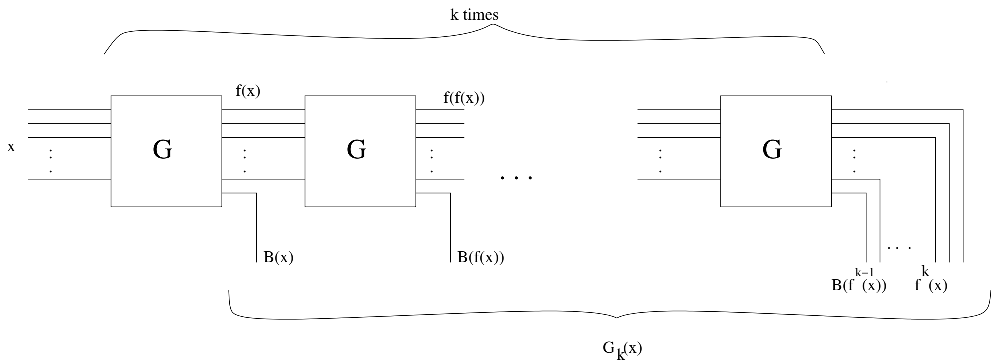

Figure 1: The Blum Micali construction

One possible approach could be the following. Consider $G : \{0,1\}^l \to \{0,1\}^{l+1}$ defined as $G(x) = x \cdot f(x)$ where $\cdot$ denotes concatenation. Since f is hard, G is a $(m^2, \frac{1}{m})$-pseudorandom generator. We could look at f as a hard core predicate for the identity permutation and mimic the Blum-Micali construction we saw in the cryptographic setting (See figure 1). However in the earlier proof, we did a bootstrap construction that worked because a permutation is hard to invert. Thus this approach cannot directly work here.
Another way to look at the Blum-Micali construction is as follows. From the seed x, we derived a sequence of strings $x, \pi(x), \pi(\pi(x)), \ldots$. Then we apply f to each of these strings to get $f(x), f(\pi(x)), f(\pi(\pi(x))), \ldots$. Similarly in this construction, from a seed s, we will generate a sequence $x_1, x_2, \ldots, x_m$, and output $f(x_1), f(x_2), \ldots, f(x_m)$. What is different in this case is the way of generating x_i's. We construct the x_i's to be independent in a limited manner so that given some values of f, it is difficult to predict the next.

3 Interlude: Combinatorial Design

To get the x_i's required above, we use a *Combinatorial design.* A *Combinatorial design* is a family of subsets $S_1, S_2, \ldots, S_m$ of $\{1, 2, \ldots, t\}$ such that

$$\begin{aligned} |S_i| &= l & \forall i \\ |S_i \cap S_j| &\le \log m & \forall i \ne j \end{aligned}$$

Lemma 2
For every l and for every $\gamma < 1$, there is a family $S_1, S_2, \ldots, S_m \subseteq \{1, 2, \ldots, t\}$ where, $t = O(\frac{l}{\gamma}), m = 2^{\gamma l}, |S_i| = l, and |S_i \cap S_j| \le \log m$. Further, the family can be computed in $2^{O(t)}$.

Proof: We describe a simple algorithm for the above. We divide the interval $[1, t]$ into l intervals of length $O(\frac{1}{\gamma})$. The algorithm basically picks the sets so that each set contains an arbitrary element from each interval. Let k be a constant to be determined later.

Input: l, γ
Output: Family $S_1, S_2, \ldots, S_m$ satisfying the above constraints.

$t := k\frac{l}{\gamma}$
Divide $\{1, 2, \ldots, t\}$ into l intervals of equal size.
Pick $S_1 \subseteq \{1, 2, \ldots, t\}$ arbitrarily such that there is exactly one element from each interval.
for $i := 2$ to m **do**
  Pick $S_i \subseteq \{1, 2, \ldots, t\}$ arbitrarily such that there is exactly one element from each interval, and further $|S_i \cap S_j| \leq \log m$ for $j = 1, 2, \ldots, i-1$.
Output $S_1, S_2, \ldots, S_m$.

This is a greedy algorithm. If it reaches the end without failing, we would be done. To prove that the algorithm works, it suffices to prove that
CLAIM:
Let $S_1, S_2, \ldots, S_i$ for $i \leq m-1$ be subsets of $\{1, 2, \ldots, t\}$ with one element per interval, where $t = k\frac{l}{\gamma}$. Then there is $S \subseteq \{1, 2, \ldots, t\}$ containing one element per interval so that $|S \cap S_j| \leq \log m$ for all $j = 1, 2, \ldots, i$.
PROOF: We show the existence of such a set S by showing a random experiment that gives such a set with non-zero probability. Construct S by choosing, for each interval, one element uniformly at random from the interval. Now for a fixed j, consider the random variable $|S \cap S_j|$. This is the sum of l independent $0/1$ random variables each with expectation $\frac{l}{t}$. Thus

$$E[|S \cap S_j|] = l\frac{l}{t} = \frac{\gamma l}{k} = \frac{\log m}{k}$$

Thus for an appropriate k the probability that $|S \cap S_j| \geq \log m$ can be made strictly smaller than $\frac{1}{m}$ (Chernoff bounds). Thus with a non-zero probability, $|S \cap S_j| \geq \log m$ for all j. $\square$
This shows that the algorithm works.

Further, we can verify that the algorithm has the required time complexity. $\square$

4 Construction

The combinatorial design above is used with the following parameters. We set $l = \frac{2\log m}{\epsilon}$ and $t = O(\frac{\log m}{\epsilon^2})$ (with $\gamma = \frac{\epsilon}{2}$). The construction of the combinatorial design then works in time $m^{O(\frac{1}{\epsilon^2})}$.
This combinatorial design gives us a family of sets with a small intersection. For a string $s \in \{0,1\}^t$ and $S_i \subseteq \{1, 2, \ldots, t\}, |S_i| = l$, define $s|_{S_i}$ to be the string of length l obtained by concatenating the bits in s indexed by S_i. We take $x_i = s|_{S_i}$ in our construction. Thus S_i's give us masks for the seed s(see figure 2). More formally, the generator is defined as

$$NW_{f,(S_1,S_2,\ldots,S_m)} : \{0,1\}^t \to \{0,1\}^m$$

$$NW_{f,(S_1,S_2,\ldots,S_m)}(s) = f(s|_{S_1})f(s|_{S_2})\ldots f(s|_{S_m})$$

Note that for $i \neq j$, $f(s|_{S_i})$ and $f(s|_{S_j})$ are almost independent in the sense that a fraction $(1-\epsilon)$ of the inputs to f are independent. This novel notion of limited independence helps us in proving that this construction gives a pseudorandom generator.

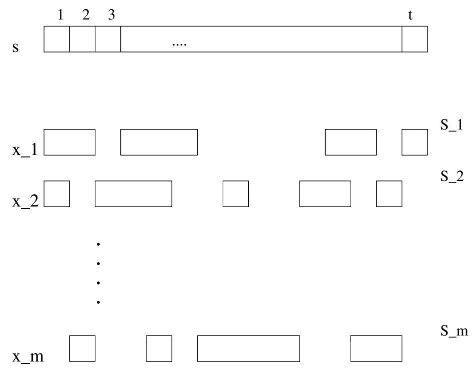

Figure 2: Getting x_i's from seed s and sets from a combinatorial design as masks

5 Proof

We now show that the construction is a pseudorandom generator. We use proof by contradiction. We show the following generalization of the contrapositive of theorem 1.

Theorem 3
Suppose that there is a circuit C of size S such that

$$\left|\mathbf{Pr}_z[C(NW_{f,(S_1,S_2,\dots,S_m)}(z)) = 1] - \mathbf{Pr}_r[C(r) = 1]\right| > \delta$$

then there is a circuit C' of size $S + O(m^2)$ such that

$$\mathbf{Pr}_x[C('x) = f(x)] \geq \frac{1}{2} + \frac{\delta}{m}$$

Proof: Without loss of generality, we can assume that

$$\mathbf{Pr}_z[C(NW_{f,(S_1,S_2,\dots,S_m)}(z)) = 1] - \mathbf{Pr}_r[C(r) = 1] > \delta \tag{1}$$

Now we use a hybrid argument as in the Blum-Micali generator. We define a sequence of distributions such that the first one is the output of the generator, the last one is a random input, and consecutive distributions are close to each other.

$$\begin{array}{lllllllll}
H_m & = & f(z|_{S_1})f(z|_{S_2})\dots f(z|_{S_i})f(z|_{S_{i+1}}) & \dots f(z|_{S_{m-1}})f(z|_{S_m}) & & & & = NW(z) \text{ for random } z \\
H_{m-1} & = & f(z|_{S_1})f(z|_{S_2})\dots f(z|_{S_i})f(z|_{S_{i+1}}) & \dots f(z|_{S_{m-1}}) & r_m & & & \\
\vdots & & & & & & & \\
H_i & = & f(z|_{S_1})f(z|_{S_2})\dots f(z|_{S_i}) & r_{i+1} & \dots & r_{m-1} & r_m & \\
H_{i-1} & = & f(z|_{S_1})f(z|_{S_2})\dots \quad r_i & r_{i+1} & \dots & r_{m-1} & r_m & \\
\vdots & & & & & & & \\
H_1 & = & f(z|_{S_1}) \quad r_2 \quad \dots \quad r_i & r_{i+1} & \dots & r_{m-1} & r_m & \\
H_0 & = & r_1 \quad r_2 \quad \dots \quad r_i & r_{i+1} & \dots & r_{m-1} & r_m & = \text{random}
\end{array}$$

So (1) says that

$$\mathbf{Pr}[C(H_m) = 1] - \mathbf{Pr}[C(H_0) = 1] > \delta$$

Therefore, there exists an i such that

$$\mathbf{Pr}[C(H_i) = 1] - \mathbf{Pr}[C(H_{i-1}) = 1] > \frac{\delta}{m} \tag{2}$$

Consider a string of length t as divided into x and y of length l and $(t-l)$ respectively, where x represents the bits indexed by S_i and y represents the remaining bits. For $j = 1, 2, \ldots, i$, define

$$f_j(x, y) = f(z|_{S_j})$$

where z is a t-bit string defined by the constraints $z|_{S_i} = x$ and $z|_{S_i^c} = y$. In other words, z is obtained by putting x in bits indexed by S_i and y in the other bits. Note that under this definition, $f_i(x, y) = f(x)$.
With this notation, (2) becomes

$$\begin{aligned} &\mathbf{Pr}[C(f_1(x,y)f_2(x,y)\ldots f_{i-1}(x,y)f(x)r_{i+1}\ldots r_m) = 1] \\ &-\mathbf{Pr}[C(f_1(x,y)f_2(x,y)\ldots f_{i-1}(x,y)r_ir_{i+1}\ldots r_m) = 1] \quad > \quad \frac{\delta}{m} \end{aligned}$$

We now define an algorithm that predicts f on an input x with a non-negligible advantage.

Algorithm A
Input: x
Output: $f(x)$

Pick at random $y \in \{0,1\}^{t-l}, b \in \{0,1\}, r_{i+1}, \ldots, r_m \in \{0,1\}$
Compute $C(f_1(x,y)f_2(x,y)\ldots f_{i-1}(x,y)br_{i+1}\ldots r_m)$
if C returns 0
  then return b
  else return $1-b$

It is easy to see that

$$\mathbf{Pr}_{x,y,b,r_{i+1},\ldots,r_m}[A(x) = f(x)] > \frac{1}{2} + \frac{\delta}{m}$$

However, note that in order to compute f once, A needs to compute f on $(i-1)$ inputs. We now fix this. First note that

$$\mathbf{Pr}_{x,y,b,r_{i+1},\ldots,r_m}[A(x) = f(x)] = \mathbf{E}_{y,b,r_{i+1},\ldots,r_m}[\mathbf{Pr}_x[A(x) = f(x)]]$$

Hence there must be fixed values of $y, b, r_{i+1}, \ldots, r_m$ that can be hardwired into a circuit C' for A such that

$$\mathbf{Pr}_x[C'(x) = f(x)] > \frac{1}{2} + \frac{\delta}{m}$$

Note that now with y fixed to a particular value y_0, $f_j(x, y_0)$ is only a function of x. Moreover, it depends on at most $\log m$ bits of x, since $|S_i \cap S_j| \leq \log m$. Since any function with n input bits has a circuit of size 2^n, $f_j(x, y_0)$ can be computed by a circuit of size $2^{\log m} = m$. Thus C' can be implemented using $O(m^2) + \text{size}(C)$ gates. $\square$