CS 6810: Theory of Computing **Spring 2026**

Lecture 16: March 19, 2026

Lecturer: Eshan Chattopadhyay *Scribe: Jiucheng Dai*

1 Nisan-Wigderson Pseudorandom Generator (NW PRG) [NW94]

Last class we proved that if there exists a function $f : \{0,1\}^n \to \{0,1\}$ that is (s,ϵ)-hard, then there exists a PRG $G : \{0,1\}^n \to \{0,1\}^{n+1}$ that is $(s-1,\epsilon)$-pseudorandom.

This is not exactly what we want, it is a PRG that takes a n-bit seed into $n+1$ bits; we want $c \log n$ bit seed into n bits. The following result shows a way to build a PRG that takes $\text{poly}(\log n)$-bit seed and output n bits. It will not be too hard to improve this construction to achieve desired performance.

Theorem 1.1. *Suppose $f : \{0,1\}^n \to \{0,1\}$ is (s,ϵ)-hard, and $f \in \text{DTIME}(2^{O(n)})$, then there exists a (s',ϵ')-PRG $G : \{0,1\}^{r(n)} \to \{0,1\}^m$ where $s' = s - O(m \cdot 2^{\frac{n}{\log n}})$, $\epsilon' = O(m\epsilon)$.*

Remark 1.2. *We will prove a particular case of the theorem: we think of $s = 2^{\delta n}, \epsilon = 2^{-\delta n}, m = 2^{\beta n}, r(n) = n^2$, where $\delta, \beta > 0$ are small constants. Using these parameters, if we have a randomized algorithm $\mathcal{B}$ that takes t input bits and takes $m = \text{poly}(t)$ time, then it will only generate at most m random bits so we can use our G to de-randomize $\mathcal{B}$ in time $2^{\text{poly}(\log t)}$ – this will make* BPP *a subset of* QuasiP*.*

Remark 1.3. *Here we will use G to brute force over all the seeds of length $r(n)$, and since this takes at least $2^{r(n)} = 2^{n^2}$ time, it doesn't hurt to have $f \in \text{DTIME}(2^{O(n)})$.*

Notice that Theorem 1.1 assumes the existence of a language $L \in \text{DTIME}(2^{O(n)})$ that is $(2^{\delta n}, 2^{-\delta n})$-hard for some small $\delta > 0$. But why is this a reasonable assumption? According to Professor Chattopadhyay, it's because if we assume so, it won't imply results that are likely false. Also, many people managed to weaken the assumption on f to a worst-case hardness assumption instead of the current average-case hardness assumption. For example, Russell Impagliazzo and Avi Wigderson did it in [IW97].

The strategy we use for the construction is the following: for a fixed input x of length $r = r(n)$, we pick a collection $\mathcal{S} = \{S_1, \cdots, S_m | S_i \subseteq [r], |S_i| = n\}$ where each S_i is the set of indices we pick from x. In other words we "sample" x_{S_i} from x, where x_{S_i} projects to S_i.

Definition 1.4. *For a fixed $\mathcal{S} = \{S_1, \cdots, S_m\}$ we define $\text{NW}_{f,\mathcal{S}}(x) = f(x_{S_1}) \circ \cdots \circ f(x_{S_m})$, where $a \circ b$ is the concatenation of a, b.*

Example 1.5. *If $x = 1010$, $\mathcal{S} = \{\{1,2,3\},\{2,3,4\},\{1,2,4\}\}$, $f(a,b,c) = a$, then $\text{NW}_{f,\mathcal{S}}(x) = 101$.*

Notice that the runtime of this construction is equal to $m \cdot$[time to compute f]+[time to generate $\mathcal{S}$] $= 2^{O(n)}$.

Notice that the output $f(x_{S_1}) \circ \cdots \circ f(x_{S_m})$ should behave like a uniformly random string, this means $S_1, \cdots, S_m$ should be relatively "independent" to each other. This can be described by the following definition:

Definition 1.6. *A (r,n,k) design is a collection $\mathcal{S} = \{S_1, \cdots, S_m\}$, such that*

1. *For all i, $|S_i| = n$, and*
2. *for all $i \neq j$, $|S_i \cap S_j| \leq k$.*

Example 1.7. *$\mathcal{S} = \{\{1,2,3\},\{2,3,4\},\{1,2,4\}\}$ is a $(4,3,2)$ design.*

Then we try to prove the function $\mathrm{NW}_{f,\mathcal{S}}$ we defined is (s',ϵ')-hard.

Claim 1.8. *If we define $\mathcal{D}_0 \to b_1 \circ \cdots \circ b_m$ to be the uniform distribution on $\{0,1\}^m$, and $\mathcal{D}_m \to f(x_{S_1}) \circ \cdots \circ f(x_{S_m})$ and $x \sim U_r$, the uniform distribution on $\{0,1\}^r$, then for any circuit C of size (at most) s', $|\mathbb{E}[C(\mathcal{D}_0)] - \mathbb{E}[C(\mathcal{D}_m)]| < \epsilon'$.*

Remark 1.9. *To prove this, we will give a hybrid argument (this technique is commonly used in cryptography). The intuition is to modify $\mathcal{D}_0$ gradually and slowly into $\mathcal{D}_m$, if each step of modification doesn't change much – with respect to m – then m modifications together shouldn't change much.*

Proof (to be continued in next lecture). Suppose there exists C of size s' such that

$$|\mathbb{E}[C(\mathcal{D}_0)] - \mathbb{E}[C(\mathcal{D}_m)]| \geq \epsilon'.$$

We define the distribution $\mathcal{D}_i$ as $f(x_{S_1}) \circ \cdots \circ f(x_{S_i}) \circ b_{i+1} \circ \cdots \circ b_m$ where $x \sim U_r$ and b_i are uniformly random bits independent of each other and independent of x. Then we see that

$$\begin{aligned}
\epsilon' \leq& |\mathbb{E}[C(\mathcal{D}_0)] - \mathbb{E}[C(\mathcal{D}_m)]| \\
=& |\sum_{i=0}^{m-1} \mathbb{E}[C(\mathcal{D}_i)] - \mathbb{E}[C(\mathcal{D}_{i+1})]| \\
\leq& \sum_{i=0}^{m-1} |\mathbb{E}[C(\mathcal{D}_i)] - \mathbb{E}[C(\mathcal{D}_{i+1})]|.
\end{aligned}$$

where the last step is by triangle inequality. Then we see that there exists an i such that $|\mathbb{E}[C(\mathcal{D}_i)] - \mathbb{E}[C(\mathcal{D}_{i+1})]| \geq \frac{\epsilon'}{m}$. Without the loss of generality, we can assume $\mathbb{E}[C(\mathcal{D}_i)] - \mathbb{E}[C(\mathcal{D}_{i+1})] \geq \frac{\epsilon'}{m}$ because we can always negate the circuit.

Then notice that $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ are very similar, and the only difference in distribution is at the $(i+1)$th bit. Utilizing this observation, we define the following randomized algorithm to contradict the hardness of f.

We define $\mathcal{A}$ that takes input $z \sim \{0,1\}^n$ and a bit b that is either a random bit or $b = f(z)$. Then $\mathcal{A}$ guesses which case it is.

The most intuitive way is to expand b into a m-bit long string and input it to C – specifically, we will want to sample a uniformly random x such that $x_{S_{i+1}} = z$ and then compute $f(x_{S_1}) \circ \cdots \circ f(x_{S_i}) \circ b \circ b_{i+2} \circ \cdots \circ b_m$ where $b_{i+2}, \cdots, b_m$ are uniformly random bits. Then if C can determine whether this is drawn from $\mathcal{D}_i$ or $\mathcal{D}_{i+1}$ we can use this to determine whether b is a uniformly random bit or is $f(z)$.

However, there are a couple of problems here:

- If x is drawn uniformly randomly, how can we guarantee that $x_{S_{i+1}} = z$?

  Notice that since z is drawn uniformly randomly, we can just sample $x_{\overline{S_{i+1}}}$ randomly can combine them, and the resulting x is still a uniformly random string.

- We can't compute $f(x_{S_i})$, because of the hardness assumption of f – if we could, we don't have to bother and we should just compute whether $f(z) = b$.

  Notice that $f(x_{S_1}) = f(x_{S_1 \cap S_{i+1}} \circ x_{S_1 \cap \overline{S_{i+1}}})$ (this is not technically always a concatenation, but a combination of input) and the idea is we only have to compute $f'(x') = f(x' \circ x_{S_1 \cap \overline{S_{i+1}}})$ and since we will define $\mathcal{S}$ as a (r, n, k)-design, f only takes in at most k bits, so f' can be computed in 2^k time. The reason why we only have to compute x' is that we will fix x except for the bits in S_{i+1}

- To contradict the hardness of f we need a circuit, but $\mathcal{A}$ is a randomized algorithm.

  In next lecture, we will modify the construction a little so that there won't be randomness needed.

□

CS 6810: Theory of Computing **Spring 2026**

Lecture 17: March 24, 2026

Lecturer: Eshan Chattopadhyay *Scribe: Shen Dong*

2 Nisan-Wigderson PRG Continued

We continue the proof of Theorem 1.1 from last lecture to prove that there exists a (s', ϵ')-hard PRG if a function $f : \{0,1\}^n \to \{0,1\}$ is (s, ϵ)-hard.

Remember that $f \in E$, the parameters satisfy $s = 2^{\delta n}$ and $\epsilon = 2^{-\delta n}$, where δ is a small constant; and that the seed length is $r = r(n)$. We have a (r, n, k)-design, which is a series of m sets $S_i \subseteq [r]$ with $|S_i| = n$ and for all $i \neq j, |S_i \cap S_j| \leq k$, and we define $NW_{f,\mathcal{S}}(x) = f(x_{S_1}) \circ \cdots \circ f(x_{S_m})$, which maps $\{0,1\}^r \to \{0,1\}^m$. The hybrids are defined as $D_i = f(x_{S_1}) \circ \cdots \circ f(x_{S_i}) \circ b_{i+1} \circ \cdots \circ b_m$ for every $i \in \{0, \ldots, m\}$, so D_0 means the random bits and D_m means the PRG output.

By the triangle inequality, our goal is to prove that every hybrid step is not distinguishable by circuits of size s' up to an advantage larger than ϵ'/m. This will be done by contradiction: if for some i, there exists a circuit C of size s' distinguishing with advantage $\geq \epsilon'/m$, then we claim that it's able to construct an adversary $\mathcal{A}$ that uses C to distinguish $f(z)$ with a uniform bit.

Claim 2.1. *If there exists a circuit C such that*

$$|\Pr[C(D_i) = 1] - \Pr[C(D_{i+1}) = 1]| \geq \epsilon'/m,$$

then there exists a randomized algorithm $\mathcal{A}$ such that

$$|\Pr[\mathcal{A}(z, b) = 1] - \Pr[\mathcal{A}(z, f(z)) = 1]| \geq \epsilon'/m,$$

where b is a bit from uniform distribution.

In the following proof we will omit the absolute value without loss of generality, since we can easily construct a circuit satisfying a certain inequality with negation. Just as the questions mentioned in the last lecture, this $\mathcal{A}$ will try to derandomize itself and reduce the running time of f. Here is the formal definition of $\mathcal{A}(z, b)$:

Algorithm 1 $\mathcal{A}(z, b)$

Require: $z \in \{0,1\}^n$, $b \in \{0,1\}$

1: Sample $x_{\overline{S_{i+1}}} \leftarrow \{0,1\}^{r-n}$
2: Set $x_{S_{i+1}} \leftarrow z$
3: Construct $D = f(x_{S_1}) \circ \cdots \circ f(x_{S_i}) \circ b \circ b_{i+2} \circ \cdots \circ b_m$;
4: **return** $C(D)$

$\mathcal{A}$ is asked to judge whether a bit b is a uniform bit or $f(z)$. $\mathcal{A}$ cannot compute $f(z)$ directly because f takes exponential time. So $\mathcal{A}$ constructs a D by placing the challenge bit b at the $(i+1)$-th index, asks C to distinguish it, and outputs what C says. Note that when $b = f(z)$, this D is actually D_{i+1}; else if b is uniform, then D is D_i. So $\mathcal{A}$ will achieve an advantage:

$$\Pr[\mathcal{A}(z, f(z)) = 1] - \Pr[\mathcal{A}(z, b) = 1] \geq \epsilon'/m$$

But this $\mathcal{A}$ still has randomness, it has to sample the $\{b_1, \ldots, b_i\}$ and $x_{\overline{S_{i+1}}}$ from uniform distribution. However, what we need is a circuit with bounded size, so we want to transform it into a deterministic algorithm. We use the probabilistic method to prove that $\mathcal{A}$'s randomness can be fixed.

Claim 2.2. *There exists a fixing of the randomness of $\mathcal{A}$, denoted by $\{b_1^*, \ldots, b_i^*\}$ and $x^*_{\overline{S_{i+1}}}$, such that the resulting deterministic algorithm $\mathcal{A}'(z, b) := \mathcal{A}_{b_1^*, \ldots, b_i^*, x_{\overline{S_{i+1}}}}(z, b)$ satisfies*

$$\Pr[\mathcal{A}'(z, b) = 1] - \Pr[\mathcal{A}'(z, f(z)) = 1] \geq \epsilon'/m.$$

Proof. We write the probability inequality with expectation, we average over all possible inputs and randomness, and because they are random independent of each other

$$\mathbb{E}_{z, \{b_1, \ldots, b_i\}, x_{\overline{S_{i+1}}}}[\mathcal{A}(z, U)] - \mathbb{E}_{z, \{b_1, \ldots, b_i\}, x_{\overline{S_{i+1}}}}[\mathcal{A}(z, f(z)] \geq \epsilon'/m$$

$$\mathbb{E}_z \left[\mathbb{E}_{\{b_1, \ldots, b_i\}, x_{\overline{S_{i+1}}}}[\mathcal{A}(z, U)] - \mathbb{E}_{\{b_1, \ldots, b_i\}, x_{\overline{S_{i+1}}}}[\mathcal{A}(z, f(z)] \right] \geq \epsilon'/m,$$

where U is the distribution of one uniform bit.

By the averaging argument that $\mathbb{E}[X] \geq a \Rightarrow \exists x \in X : \ x \geq a$, there exists a randomness (i.e., the bits $x^*_{\overline{S_{i+1}}}$ and $\{b_1^* \ldots, b_i^*)$ that is good enough to have

$$\mathbb{E}_z \left[\mathcal{A}_{\{b_1^*, \ldots, b_i^*\}, x^*}(z, U) - \mathcal{A}_{\{b_1^*, \ldots, b_i^*\}, x^*}(z, f(z)) \right] \geq \epsilon'/m.$$

Therefore, we hardwire this set of good randomness as advice, resulting deterministic algorithm $\mathcal{A}'$ preserves the same advantage. Now we "derandomize" the algorithm $\mathcal{A}$ while keeping it's ability to distinguish between uniform bit and f. □

Now we need to compute the circuit size for $\mathcal{A}'$. We still need to feed $f(x_{S_{i+2}}), \ldots, f(x_{S_m})$ into C, but we of course don't want to compute f completely. Luckily, for all $j \neq i+1$, we have $|S_j \cap S_{i+1}| \leq k$ by the (r, n, k)-design. Observe that in $f(x_{S_j}) = f(x_{S_j \cap S_{i+1}} \circ x_{S_j \cap \overline{S_{i+1}}})$, the size of the former part is bounded by k, and the latter part is hardwired. So for every $j \neq i+1$, we construct $f_j : \{0,1\}^{n_j} \to \{0,1\}$, $n_j = |S_j \cap S_{i+1}|$, f_j takes the bits in x_{S_j} that is not fixed by $x_{S_{i+1}} := z$ and computes $f_j(x_{S_j \cap S_{i+1}}) := f(x_{S_j \cap S_{i+1}} \circ x_{S_j \cap \overline{S_{i+1}}})$. So the input length of f_j is less than k, we just need a circuit of size 2^k to compute $f(x_{S_j})$.

We don't know what i is, but we know $i \leq m$, so we at most need to compute f for m times and feed m hardwired random bits into C. Since input z occupies n bits in the r-length seed, fixing randomness requires hardwiring $(r-n)+(m-1)$ bits and we need a sub-circuit of size $(m-1) \cdot 2^k)$ in total for all f computation. Therefore with typical parameter choice $r < m$, the circuit size of $\mathcal{A}$ in total should be $\leq 2m + m \cdot 2^k + s' < s$, $s' = s - O(m2^k)$.

One last question we care about is how may S sets we can have in a (r, n, k)-design, i.e. how many bits this $NW_{f,\mathcal{S}}$ can output. Say we set $r = n^2$, $k = \frac{\delta n}{10 \log n}$. We pick a finite field of size n, denoted by $\mathbb{F}$, and the whole seed universe we care is $\mathbb{F} \times \mathbb{F}$ (of size r). For every polynomial $p : \mathbb{F} \to \mathbb{F}$ with coefficients in $\mathbb{F}$ and degree $< k$, we construct a set $S_p = \{(z, p(z)) : z \in \mathbb{F}\}$, which contains n elements in the universe. So each polynomial p with degree $< k$ decides a valid set S_p in our collection $\mathcal{S}$, and we have at most n^k different polynomials thus $m = n^k$ different sets. We still need to ensure the intersection of S_p and S_q is less than k for $p \neq q$. Note that $S_p \cap S_q = |\{z : p(z) - q(z) = 0\}|$, this requirement is naturally fulfilled by the fundamental theorem of algebra: p, q are both polynomial with degree $< k$, so when $p \neq q$, $p - q$ is also a non-zero polynomial with degree $< k$, the number of roots to $p - q$ is at most $k - 1$, strictly less than k.

Until now, we have finished the full proof of *NW*'s PRG construction. There are still many interesting topics to explore, like how to extend the analysis to multilinear polynomial variant. Furthermore, the NW generator construction we learn so far was analyzed under the assumption that there exists a Boolean function $f : \{0,1\}^n \to \{0,1\}$ in E that is already (s, ϵ)-hard on average, where typically $s = 2^{\delta n}$ and $\epsilon = 2^{-\delta n}$. A major result of Impagliazzo and Wigderson [IW97] shows that one can start from a weaker *worst-case* hardness assumption to achieve $\mathbf{BPP} = \mathbf{P}$.

3 Interactive Proofs

We also cover a brief background on interactive proofs in this lecture.

Fix a verifier V for a language L, and require that V run in polynomial time. Then the usual definition of NP can be rewritten in proof-system language as follows: the prover sends a certificate c, and the verifier checks it deterministically in polynomial time.

Formally, if $x \in L$, then there exists an (honest) prover P such that P sends a certificate c and $V(x, c) = 1$. On the other hand, if $x \notin L$, then for every prover P and every message c, we have $V(x, c) = 0$. Thus, NP is exactly the class of languages that admit such one-message proofs.

A natural question is what happens if the prover and verifier are allowed to interact more. This leads to the notion of an interactive proof system, where the prover and the verifier may exchange multiple messages while the verifier still runs in polynomial time. A fundamental theorem [Sha92] in complexity theory states that

$$\mathbf{IP} = \mathbf{PSPACE}.$$

Acknowledgement

I want to thank Professor Eshan Chattopadhyay and Mohit Gurumukhani for the helpful discussions and additions and suggestions to this document.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[IW97] Russell Impagliazzo and Avi Wigderson. P = bpp if e requires exponential circuits: derandomizing the xor lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 220–229, New York, NY, USA, 1997. Association for Computing Machinery.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[Sha92] Adi Shamir. Ip= pspace. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.

[Wik25] Wikipedia contributors. Pseudorandom generator — Wikipedia, the free encyclopedia, 2025. [Online; accessed 19-March-2026].