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

Lecture 15: March 17, 2026

Lecturer: Eshan Chattopadhyay *Scribe: Shen Dong*

Today's main topic is hardness vs randomness. One of the central questions in derandomization is whether randomness actually adds computational power. The standard formulation is whether **BPP** = **P**. A common approach is to construct pseudorandom generators that can fool efficient algorithms, and then to use such generators to remove randomness from **BPP** computations.

1 Pseudorandom Generator

Definition 1.1. *(Pseudorandom Generator) For an $S : \mathbb{N} \to \mathbb{N}$, $G : \{0,1\}^{r(n)} \to \{0,1\}^n$ is an (S, ε)-PRG if for every circuit C of size at most $S(n)$,*

$$|\Pr[C(U_n) = 1] - \Pr[C(G(U_r)) = 1]| < \varepsilon,$$

where U_n is the uniform distribution over $\{0,1\}^n$, and $r(n)$ is the seed length of PRG G, and it grows much more slowly than n.

PRGs are important tool in derandomization. For derandomization, the dream is to have the seed length $r(n) = O(\log n)$, because then we can enumerate all $2^{r(n)}$ strings in polynomial time, going over all possible randomness.

2 Derandomization

We now do a basic derandomization argument.

Claim 2.1. *If there exists an (S, ε)-PRG G which satisfies the following properties, then **BPP** = **P**.*

1. *seed length $r(n) = O(\log n)$;*
2. *allow the distinguishing circuit to have size $S(n) = poly(n)$;*
3. *G is computable in time $poly(n)$;*
4. *ε is a constant less than $< \frac{1}{6}$, say $\frac{1}{10}$.*

Proof. Suppose $L \in$ **BPP**, then there exists an randomized TM M such that for all input x with length $|x| = n$, $\Pr_{y \sim U_\ell}[M(x, y) = L(x)] \geq \frac{2}{3}$ with $\ell = poly(n)$.

Fix the input to an x, and define a circuit $C(y) = M(x, y)$, where $C : \{0,1\}^\ell \to \{0,1\}$, and size of C is $\leq poly(n)$. By pseudorandomness, when we parameterize G's output length to be $\ell(n)$, we have

$$|\Pr[C(U_\ell) = 1] - \Pr[C(G(U_r)) = 1]| < \varepsilon,$$

Intuitively, if $x \in L$, we have

$$\begin{aligned} \Pr[C(G(U_r)) = 1] &= \Pr[M(x, G(U_r)) = 1] \\ &\geq \Pr_{y \sim U_\ell}[M(x, y) = 1] - \varepsilon \\ &\geq 2/3 - \varepsilon; \end{aligned}$$

while if $x \notin L$, then

$$\begin{aligned}\Pr[C(G(U_r)) = 1] &= \Pr[M(x, G(U_r)) = 1] \\ &\leq \Pr_{y \sim U_r}[M(x, y) = 1] + \varepsilon \\ &\leq 1/3 + \varepsilon.\end{aligned}$$

With ϵ being a constant less than $1/6$, there will be a gap to make **majority voting** work to distinguish this.

We deterministically compute $M(x, y_1), \ldots, M(x, y_R)$ for all possible seeds of G, i.e. $s \in \{0,1\}^r$, so $R = 2^r$, and y_i is generated with the i-th seed s_i using G. Then output the majority of all outputs, the running time of this process is $R \cdot (t(M) + t(G)) = poly(n)$. □

As shown in the above example, we run G over all the seeds to make a randomized process deterministic. So G has to be extremely efficient.

3 Hardness assumptions

We now connect PRGs with circuit lower bounds. A typical technique is to start with some hardness assumption, construct PRG and use it to do derandomization. The intuition is that one starts from sufficiently hard functions and builds long-stretch PRGs that fool polynomial-size circuits. Therefore, if there is a function which is hard to approximate for some circuit class, then we can derandomize **BPP** to deterministic complexity class. In the following part we only take one step into this, showing how to build PRGs from hardness.

Definition 3.1. *(Average-case Hardness) A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is (S, ε)-hard if for all circuit of size less than S, $\Pr_{x \sim U_n}[f(x) = C(x)] \leq \frac{1}{2} + \varepsilon$.*

Claim 3.2. *Given such function f which is (S, ε)-hard, define $G(x) = (x, f(x)) \in \{0,1\}^{n+1}$. Then G is a $(S - 1, \varepsilon)$-PRG.*

Proof. Suppose there is a C of size $S - 1$ that can distinguish between G's output and uniform random, then there will be

$$\Pr_{x \sim U_n}[C(x, f(x)) = 1] - \Pr_{x \sim U_n, b \sim U_1}[C(x, b) = 1] \geq \varepsilon.$$

Note that we can always get a C satisfying this without an absolute value mark, because if a circuit's advantage against uniform strings are larger, we can just negate its output to get such a C.) We can construct a circuit A that predicts $f(x)$: A pick a random bit c, if $C(x, c) = 1$, then

output c; else if $C(x, c) = 0$, output $1 - c$. One can show that

$$\begin{aligned}
&\Pr_{x\sim U_n}[A(x) = f(x)] \\
&= \Pr[A(x) = f(x) \mid c = f(x)] \cdot \Pr[c = f(x)] + \Pr[A(x) = f(x) \mid c = 1 - f(x)] \cdot \Pr[c = 1 - f(x)] \\
&= \Pr[C(x, c) = 1] \cdot \Pr[c = f(x)] + \Pr[C(x, c) = 0] \cdot \Pr[c = 1 - f(x)] \\
&= \frac{1}{2}\left(\Pr[C(x, f(x)) = 1] + \Pr\left[C(x, 1 - f(x)) = 0\right]\right) \\
&= \frac{1}{2}\left(\Pr[C(x, f(x)) = 1] + 1 - \Pr\left[C(x, 1 - f(x)) = 1\right]\right) \\
&= \frac{1}{2} + \frac{1}{2}(\Pr[C(x, f(x)) = 1] - \Pr\left[C(x, 1 - f(x)) = 1\right]) \\
&= \frac{1}{2} + \frac{1}{2}(2\Pr[C(x, f(x)) = 1] - (\Pr[C(x, f(x)) = 1] + \Pr\left[C(x, 1 - f(x)) = 1\right])) \\
&= \frac{1}{2} + \frac{1}{2}(2\Pr[C(x, f(x)) = 1] - (\Pr[C(x, c') = 1] + \Pr\left[C(x, 1 - c') = 1\right])) \\
&= \frac{1}{2} + \frac{1}{2}(2\Pr[C(G(x)) = 1] - 2\Pr[C(U_{n+1}) = 1]) \\
&\geq \frac{1}{2} + \varepsilon,
\end{aligned}$$

where the second equation comes from the construction of $A(x)$, the third equation comes from the fact that c is a random coin, and in the seventh equation, we introduce another random coin c', because we talk about both $f(x)$ and $1 - f(x)$ case in the last term, which is equivalent to flipping a random coin. so in the last equation we get (x, c') U_{n+1}. Therefore, A is a circuit that can approximate f better than ε, and A actually runs $C(x, c) \oplus c$, so its size will be $S - 1 + 1 = S$, which is enough to contradict the hardness of f. Therefore, C should not exist, G is a $(S - 1, \varepsilon)$-PRG. □

The construction $G(x) = (x, f(x))$ stretches the input by just one bit and gives us a taste of the proof style though by itself it's not enough to derandomize **BPP**. But constructions with short enough seeds and large enough stretch will enable all the seeds to be enumerated and replace randomness with deterministic computation.

Remark 3.3. *Nisan and Wigderson showed that we can construct PRGs capable of efficiently simulating randomized algorithms from average-case hard functions [2]. And there is a stronger result from Impagliazzo and Wigderson that even worst-case hardness assumption can still be used to construct PRG and go towards* $\boldsymbol{BPP = P}$ *[1].*

References

[1] 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. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 220–229. ISBN: 0897918886. DOI: 10.1145/258533.258590. URL: https://doi.org/10.1145/258533.258590.

[2] N. Nisan and A. Wigderson. "Hardness vs. randomness". In: *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*. 1988, pp. 2–11. DOI: 10.1109/SFCS.1988.21916.