CS761 Derandomization and Pseudorandomness **2022-23 Sem I**

Lecture 13: 27-09-2022

Scribe: Amit Rajaraman *Lecturer: Rohit Gurjar*

A general question one can ask is this: for any polynomial-time randomized algorithm, is it possible to derandomize it to get a polynomial-time deterministic algorithm doing the same job?
It has been shown that given a "hard" function, one can construct very good pseudorandom bits. However, no explicit hard functions are known.

Consider the notion of *worst case hardness*. For example, if we can show for some language that no algorithm that runs in $O(n^{10})$ time can compute the output correctly on all inputs, then the language is hard in some sense.

We also have the notion of *average case hardness*. Here, if we can show for some language that no algorithm that runs in $O(n^{10})$ time can compute the output correctly on more than $3/4$ of the inputs, then the language is hard in some sense. It is not too difficult to see that average case hardness is a stronger notion than worst case hardness.

Problems that are average case hard yield good pseudorandom generators. Another question of concern is converting worst case hardness to average case hardness, which is done through error correcting codes. This will be discussed in subsequent lectures.

Definition 13.1 (Randomized polynomial time (RP))**.** *A language L is said to be in* RP *if there is a randomized algorithm $\mathcal{A}$ running in polynomial time such that*

1. *for $x \in L$,*
$$\Pr_r [\mathcal{A}(x, r) = \mathsf{yes}] \geq \frac{1}{2}.$$
2. *for $x \notin L$,*
$$\Pr_r [\mathcal{A}(x, r) = \mathsf{yes}] = 0.$$

In other words, the problems in class RP have probabilistic algorithms that make one-sided error. For example, we had discussed an algorithm for determining (s, t)-connectivity in a graph. The algorithm would do a random walk starting from s and says yes if it sees t in a bounded number of steps, otherwise says not connected. Observe that if s and t are not connected then the algorithm will correctly say that with probability 1. But, if they are connected then the algorithm may give the wrong answer (not connected) with some probability. That's what we mean by one-sided error.

We can also define a complexity class for algorithms that are allowed to make two-sided errors.

Definition 13.2 (Bounded-error probabilistic polynomial time (BPP))**.** *A language L is said to be in* BPP *if there is a randomized algorithm $\mathcal{A}$ running in polynomial time such that*

1. *for $x \in L$,*
$$\Pr_r [\mathcal{A}(x, r) = \mathsf{yes}] \geq \frac{2}{3}.$$
2. *for $x \notin L$,*
$$\Pr_r [\mathcal{A}(x, r) = \mathsf{yes}] \leq \frac{1}{3}.$$

Here, the choice of numbers $2/3$ and $1/3$ was arbitrary. We can take any two numbers where there is a constant gap, for example, $1/4$ and $1/6$.

Next, let us look at pseudorandom distributions. The goal of these is to find some universal set of random bits which we can substitute in place of the (ideally) uniformly random bits being used in an algorithm.
Denote by U_m the uniform distribution on $\{0, 1\}^m$.

The idea is that a distribution D is pseudorandom (with respect to $\mathcal{A}$) if for the given algorithm $\mathcal{A}$, for all inputs x,

$$\left| \Pr_{r \sim U_m} [\mathcal{A}(x, r) = \mathsf{yes}] - \Pr_{r \sim D} [\mathcal{A}(x, r) = \mathsf{yes}] \right| \leq \epsilon.$$

One convenient way to think about algorithms *with input* is circuits.
In fact, any deterministic algorithm can be viewed as a family $(C_n)_{n \geq 1}$ of circuits, with C_n computing the output correctly if the input is of size n. We can view a randomized algorithm as a deterministic one with two inputs, x and r. The circuit then has $n + m$ input leaves, where n is the size of the input x and m is the number of random bits r. If we fix the x part of the input, we get a circuit in the remaining input, namely $r_1, \ldots, r_m$. We will call a given distribution pseudorandom if a small circuit cannot distinguish it from the uniform distribution.

Definition 13.3 (Pseudorandom distribution)**.** *A distribution D on $\{0,1\}^m$ is called* (S, ϵ)-pseudorandom *if for any circuit C on m input gates and size at most S,*

$$\left| \Pr_{r \sim U_m} [C(r) = 1] - \Pr_{r \sim D} [C(r) = 1] \right| \leq \epsilon.$$

Definition 13.4 (Pseudorandom generator (PRG))**.** *Let $G = (G_\ell)_{\ell \in \mathbb{N}}$ be a family of functions, where $G_\ell : \{0,1\}^\ell \to \{0,1\}^{m(\ell)}$ (for some appropriate increasing function m). G is said to be a* $m(\ell)$-pseudorandom generator *if for each ℓ,*

(a) For a given $r \in \{0,1\}^\ell$, $G_\ell(r)$ can be computed in $2^{O(\ell)}$ time, and

(b) the distribution $G(U_\ell)$, that is the distribution over $\{0,1\}^{m(\ell)}$ obtained by taking a uniformly random element s from $\{0,1\}^\ell$ and outputting $G(s)$, is $(m(\ell)^3, 1/10)$-pseudorandom.

The existence of the above PRG would imply that any randomized algorithm in BPP using m random bits and running time m^3 can be simulated by a deterministic algorithm with running time $2^{O(\ell)} m^3$. We merely run the algorithm on $G(r)$ for all $r \in \{0,1\}^\ell$, and output whatever answer (yes or no) occurs more frequently.
If we want to completely derandomize our randomized polynomial-time algorithm to get a deterministic polynomial time algorithm, we will need a PRG with $m(\ell) = 2^{O(\ell)}$. That is, if a pseudorandom generator with seed length $\ell = O(\log m)$ exists, then $\mathsf{BPP} = \mathsf{P}$. When this is true, we say that the PRG has *exponential stretch*.

Our goal in the next few lectures will be to show that "circuit lower bounds" imply the existence of pseudorandom generators. First we give an argument which shows why we can expect such PRGs to exist.

Theorem 13.5. *There exists a "PRG" with exponential stretch which satisfies only (b) in the definition.*

Proof. Choose a random G – for each $s \in \{0,1\}^\ell$, set $G(s)$ to be a uniformly random string in $\{0,1\}^m$.
Fix a circuit C of size at most m^3. Suppose that $\Pr_{r \sim U_m}[C(r) = 1] = p$, and let

$$B_C \coloneqq \{r \in \{0,1\}^m : C(r) = 1\}.$$

Note that the random variable

$$X_C \coloneqq \left| \left\{ s \in \{0,1\}^\ell : G(s) \in B_C \right\} \right|$$

is the sum of 2^ℓ Bernoulli random variables equal to 1 with probability p. By the Chernoff bound,

$$\Pr_G \left[|X_C - \mathbb{E}[X_C]| > \epsilon 2^\ell \right] \leq 2e^{-(\epsilon 2^\ell)^2 / 3\mathbb{E}[X_C]} = 2e^{-(\epsilon 2^\ell)^2 / (3p \cdot 2^\ell)} \leq 2e^{-\epsilon^2 2^\ell / 3}.$$

Now, the number of circuits with size at most m^3 is bounded from above by 2^{3m^3}. An application of the union bound yields that

$$\Pr_G[\text{for some } C \text{ of size at most } m^3,\ |X_C - \mathbb{E}[X_C]| > \epsilon 2^\ell] \leq e^{-(\epsilon^2 2^\ell / 3) + (\ln 2)(3m^3 + 1)}.$$

For $\epsilon = 1/10$, $\ell = 4 \log m$, and sufficiently large m, the RHS is less than 1, so there exists some G that satisfies (b) in the definition of a PRG. $\square$

Finally, is it possible to get a PRG with even larger (than $2^{O(\ell)}$) stretch? The answer is no. Observe that the distribution $G(U_\ell)$ is supported on 2^ℓ strings. Hence, one can construct a circuit of size $m \times 2^\ell$ that accepts exactly those strings which are in the support of $G(U_\ell)$. This circuit distinguishes $G(U_\ell)$ from the uniform distribution U_m. In other words we cannot hope to fool all circuits of size $m \times 2^\ell$. That means, according to the above definition of a PRG, we must have $m \leq 2^{\ell/2}$. Hence, at best we can have an exponential stretch in a PRG.