CS 6815: Pseudorandomness and Combinatorial Constructions **Fall 2022**

Lecture 14: Oct 6, 2022

Lecturer: Eshan Chattopadhyay *Scribe: Ricky Shapley*

1 Finishing the Nisan-Wigderson construction

Recall the Nisan-Wigderson PRG we described in the previous class. We stated that if there is a language $L \subseteq \{0,1\}^*$ which is (s,ϵ)-hard, then there is a PRG $\{G_n\}_{n\geq 0}$ mapping $\{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$ that is (s',ϵ')-pseudorandom, where $s' = s - O(m2^k)$ and $\epsilon' = m \cdot \epsilon$.

For the construction to work, we need an explicit (n,k)-design. That is, we need to be able to construct sets $S_1, \ldots, S_m \subseteq [r]$ with $|S_i| = m$ for $i \in [m]$ and $|S_i \cap S_j| \leq k$ for $i \neq j \in [m]$.

1.1 Explicit construction of (n,k)-design

We pick a field $\mathbb{F}$ of size n, and define a set for every polynomial of degree at most k. For such polynomials p, let

$$S_p = \{(x, p(x)) : x \in \mathbb{F}\}.$$

Notice that for $p \neq p'$,

$$\left|S_p \cap S_{p'}\right| - \left|\{x \in \mathbb{F} : p(x) = p'(x)\}\right| \leq k.$$

Then there are something like n^{k+1} such polynomials (and therefore just as many sets) since there are $k+1$ monomials and n choices for each coefficient. So we can have a (n,k)-design on a universe of size $r(n) = n^2$ with $m(n) = n^k$ sets. (We can just exclude the extra sets from our design for simplicity here.)

Now we choose some parameters. We will let $s(n) = 2^{\delta n}$ and $\epsilon(n) = 2^{-\delta n}$ for some constant δ. Then, it makes sense to choose $k = \frac{\delta n}{10 \log n}$, which gives us that $m(n) = 2^{\delta n/10}$. Using this in the Nisan-Wigderson PRG means the following:

Suppose there is some $\delta > 0$ such that the language L is $(2^{\delta n}, \frac{1}{10}2^{\delta n})$-hard, then there is a PRG $G : \{0,1\}^{n^2} \to \{0,1\}^{2^{\frac{\delta n}{10}}}$ which is $(\frac{2^{\delta n}}{2}, 2^{-\frac{9\delta n}{10}})$-pseudorandom. So if we make a strong enough assumption that exponentially large circuits cannot approximate our language, then we can find a PRG to fool circuits of size m, that only requires $O(\log m)$ many random bits. This in turn would imply that $\mathbf{BPP} = \mathbf{P}$.

1.2 Restrictions on L

One extra note is that we should make some restrictions on what our choice of language L can be, in particular it should be in $\mathbf{DTIME}(T(n))$ for some reasonable choice of $T(n)$. Remember that if L is in $\mathbf{DTIME}(T(n))$, then it can be computed by circuits of size $T(n)\log(T(n))$. So if, for example, we ask $T(n)$ to be polynomial in n, then our above theorem is meaningless as we can compute our language with some polynomially sized circuit.

In the construction of the PRG, we evaluate the m inputs to check if they are in L, and m is exponential in n. So letting $T(n) = 2^{O(n)}$ is fine because it doesn't affect the asymptotic runtime.

This is a difference from cryptography, where we would not allow the PRG to be exponential n. But since we will use our PRGs to derandomize by running over all seeds, this will take time that is exponential in n but still polynomial in m, which is what we desire.

1.3 Relation to circuit lower bounds

In our theorem, we supposed that there was a language that some class of circuits could not approximate. We can express this assumption as a circuit lower bound. Restating, we supposed there is some language in $\mathbf{E} = \mathbf{DTIME}(\mathbf{2^{O(n)}})$, and L is $2^{\delta n}$-hard. This is a circuit lower bound against the complexity class $\mathbf{E}$.

We know that $\mathbf{P}$ in $\mathbf{P/poly}$ so suppose there exists L in $\mathbf{NP}$ and L is slightly superpolynomial (L is $n^{\omega(n)}$-hard) and cannot be computed by polynomially sized circuits. (This is even weaker than saying it cannot be approximated.) This would imply $\mathbf{P} \neq \mathbf{NP}$ because then $L \in \mathbf{NP}$, but not $\mathbf{P}$. But the best known circuit lower bound for languages in $\mathbf{NP}$ is $3.1n$, which is quite far away from the assumptions we have made to derive our derandomization results.

2 Randomness Extractors

Lots of algorithms assume you have random bits, but in reality, we only have bits from some distribution D on $\{0,1\}^n$ that contains randomness. So how do we generate pure random bits?

We consider an example of a simple source of randomness.

Example 2.1 (von Neumann Extractor)**.** *Given $x_1, \ldots, x_n$ where $x_i \sim$Bernoulli(p), with $\delta < p < 1-\delta$, can we extract one bit $y(x)$ such that y is almost random? That is, $|Ey - 1/2| \leq \epsilon$.*

There are several ways of doing this. For example $\bigoplus_{i=1}^n x_i$ works. (In fact, the exact probabilities of each outcome are given by the first row of $\begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}^n$, which converges to the uniform distribution exponentially fast.)

Another solution is to consider the bits of x in pairs, and if x_{2i-1} and x_{2i} differ, we use $y(x) = x_{2i-1}$, otherwise we repeat with the next pair. It is clear that if the bits differ then 0 and 1 are outputted with equal probability. And the probability of having to repeat is $1 - 2p(1-p)$, so the probability of repeating t times is

$$(1 - 2p(1-p))^t \leq e^{-2p(1-p)t}\epsilon$$

where we bound the error by the likelihood of repeating all t times. This implies to get an error of at most ϵ, we should flip at least

$$t = \frac{1}{2\delta(1-\delta)} \log(1/\epsilon)$$

bits. (Again, this converges to the uniform distribution exponentially fast in the number of bits in x).

Definition 2.2. *Informally, an extractor is a deterministic algorithm that gets a sample from a defective source and outputs almost uniform bits.*

Ideally, we want to know if we can have randomized algorithms that still work, even with defective randomness.

2.1 Modeling defective randomness

What is the right way of modeling a weak source? There has been a lot of work on this question, and by now the standard way is using min-entropy that we now define.

Definition 2.3. *Given some distribution X over $\{0,1\}^n$, the min-entropy $H_\infty(X)$ is*

$$H_\infty(X) = \min_{x \in sup(X)} \left\{ \log \left(\frac{1}{P[X = x]} \right) \right\}.$$

Equivalently, this implies that if $H_\infty(X) \geq k$, then $P[X = x] \leq 2^{-k}$ for all $x \in X$. Conceptually, it is a measurement of the maximum weighted element in X.

Example 2.4. *Take any $S \subseteq \{0,1\}^n$ with $|S| = 2^k$. Let X be uniform on S. Then $H_\infty(X) = k$.*

In fact, though we do not prove it here, it is sufficient to just consider this kind of flat distributions where all elements in the support of X are equally likely.