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

Lecture 17: 10-10-22

Scribe: Punna Hitesh Kumar *Lecturer: Rohit Gurjar*

From worst case hardness to average case hardness

In the last few lectures, we have seen that a function which is hard in average case can be used to design a pseudorandom generator. We know that there exist functions which are exponentially hard in worst case. But, do we even know existence of functions which are hard on average? In the next few lectures, we will see that starting from a worst case hard function, we can construct an average case hard function. This is known as amplifying hardness. That would mean that PRGs can be constructed from worst case hard functions. The key idea that will be used is error correcting codes.

Definition 17.1. *For a number $\rho \in (0,1)$ and function : $f\colon \{0,1\}^n \to \{0,1\}$, we define $H^{\rho}_{avg}(f)$ to be the largest number S such that for any circuit C of size S,*

$$\Pr_{x\in\{0,1\}^n}[f(x) = C(x)] < \rho.$$

Note that the worst case hardness can be defined as $H_{wrs}(f) = H^1_{avg}(f)$. In other words, the largest size of circuits that do not compute f correctly on all inputs. Recall that we had defined average case hardness $H_{avg}(f)$ as the largest S such that for any circuit of size S

$$Pr_{x\in\{0,1\}^n}[C(x) = f(x)] < 1/2 + 1/S.$$

Intuition. For any worst case hard function $f\colon \{0,1\}^n \to \{0,1\}$, consider the truth table of the function which can be viewed as a 2^n bit string. The idea is to apply an error correcting code on this 2^n bit string and obtain a possibly larger, say 2^{2n} bit string. Then view this new string as a function $f'\colon \{0,1\}^{2n} \to \{0,1\}$. We would like to claim that f' is hard on average case. Worst case hardness of f means that for any small enough circuit C, the truth tables of f and C will disagree on at least one bit. If we apply an error correcting codes on both the truth tables then we will get that the two encodings disagree on a significant fraction of bits. In other words, f' is computed incorrectly on a significant fraction of inputs by the 'circuit' C' (which is obtained by encoding of C). However, we want to show a stronger statement for any small enough circuit, f' is computed incorrectly on a significant fraction of inputs. Let's try to argue this formally.

Let, E be the encoding function of a code where $E : \{0,1\}^{2^n} \to \{0,1\}^{2^{2n}}$. Let's say the relative distance of the code is $\Delta = 1/4$. The new function $f' : \{0,1\}^{2n} \to \{0,1\}$ is defined such that

$$\text{truth-table}(f') = E(\text{truth-table}(f)).$$

Consider the contrapositive statement. Suppose there is a circuit B of size S on $2n$-bit inputs, such that, $B(y)$ and $f(y)$ agree on a large set (say 0.9 fraction of inputs) i.e., $Pr_{y\in\{0,1\}^{2n}}[B(y) = f(y)] = 0.9$. Then we need to show there is a circuit C of size poly(S) such that $\Pr_{x\in\{0,1\}^n}[C(x) = f(x)] = 1$.

How can we construct the circuit C from B. From the property of the error correcting codes, we know that given any string in $\{0,1\}^{2^{2n}}$ which agrees with the codeword truth-table(f') on a large fraction, we can use the decoding algorithm to recover the 'message', which is the truth-table(f). Once we have the truth-table for f, we can try to construct a circuit for f. The problem with this approach is that to decode, one needs the whole 2^{2n} bit string (truth-table of B). And that means the circuit for f will be at least 2^{2n} times the size of B. To fix this issue, what we need is local decoding.

Local decoding. Instead of going through the entire encoded word, we will get to see only a few randomly chosen locations in the word, and from that we will need to decode some part of the message.

Definition 17.2 (Local decoding)**.** *Let E be an encoding algorithm, $E : \{0,1\}^N \to \{0,1\}^M$. Encoding should be doable in poly(N) time. A local decoder for handling ρ errors is an algorithm D, that given a random access to a string $y \in \{0,1\}^M$ such that $\Delta(y, E(x)) \leq \rho$ for some $x \in \{0,1\}^N$, and an index $j \in [1, N]$, runs in time* $\mathrm{poly}(\log(M))$ *and outputs x_j with probability at least 2/3.*

Note that in particular, the above definition implies that a local decoding algorithm can see only $\mathrm{poly}(\log(M))$ locations (can be randomly chosen) of the given codeword. Now, let us see how local decoding can help with amplifying hardness.

Theorem 17.3. *Let D be a local decoder for handling ρ errors for encoding function $E : \{0,1\}^N \to \{0,1\}^M$. Suppose there is a function $f : \{0,1\}^n \to \{0,1\}$ and $f \in E$ (exponential time computable) and $H_{wrs}(f) > S(n)$. Then there exist $\epsilon > 0$ and there is $f' \in E$ (exponential time computable) on m-bits, such that*

$$H^{1-\rho}_{avg}(f') \geq S(\epsilon m)/\operatorname{poly}(m).$$

Proof. Let $N = 2^n$ and $M = 2^m$. Define f' such that

$$\text{truth-table}(f') = E(\text{truth-table}(f)).$$

Let B be a circuit of size T such that $Pr_x[B(x) = f'(x)] \geq 1 - \rho$. Equivalently,

$$\Delta(\text{truth-table}(B), \text{truth-table}(f')) \leq \rho.$$

Hence, we can use the decoding algorithm D on truth-table(B) to recover a particular bit in truth-table(f). Each time we compute $B(y)$ for some y, we spend time T. Since D runs in $\operatorname{poly}(\log M))$ time, the size of the circuit C computing f will be

$$T \operatorname{poly}(\log M)) = T \operatorname{poly}(m).$$

Note that $m = O(n)$. Now, using the hardness assumption on f will give us the desired result.

One important point to note is that the decoder algorithm D is randomized. To get a deterministic circuit for f, we can simply say that there exist one good choice of random bits that ensures correct computation, and just hardcode this choice in the circuit. But, that would mean we need the same choice of random bits to work for every input $x \in \{0,1\}$. To get this, first one needs to boost the success probability of the decoder to very high (more than $1 - 1/2^n$). And then apply union bound, to argue that there is a choice of random bits that works for every x. □