Lecture

Today, we like to come back to a topic we discussed before: pseudo-random generators. Before, we saw their equivalence to 1-way functions, and their power in cryptographic schemes. Here, we are interested in using pseudo-random generators to deterministically simulate probabilistic algorithms.

Definition 1 *A deterministic procedure is a pseudo random generator G if the following holds: $G : \{0,1\}^t \to \{0,1\}^m, m > t$, is an (s, ϵ)- pseudo random generator if for every circuit C of size $\leq S$,*

$$|\mathbf{Pr} r C(r) = 1 - \mathbf{Pr} x C(G(x)) = 1| \leq \epsilon$$

Suppose we have a probabilistic polynomial-time algorithm, A. That is, A runs in time n^c on input of length n, and gives the right answer with probability $> 3/4$. Suppose for every n we have a generator mapping n to $G_n : \{0,1\}^{T(n)} \to \{0,1\}^{n^c}$ which is $(n^c, 1/10)$ pseudo random computable in time $T(n)$. Using the fact that every algorithm that runs in time T can be simulated by a circuit of size T^2,then for every fixed x, the same computation can be simulated by a circuit C_A that outputs the right answer for x with probability $\geq 3/4 - 1/10 > 1/2$, where A given x and gives out the right answer with probability $3/4$. Therefore, A and C_A have agreement within $1/10$.

As long as the pseudo random generator's probability is $\alpha/2$, where α is the probability by which the algorithm A is bounded away from $1/2$ then we can replace the pseudo random generator with random string and get probability $> 1/2$. Hence, for every x, $\mathbf{Pr} A(x), G(s) = \text{right answer for } x > 1/2$.

The right answer for x can be found in time $2^{t(n)}(T(n) + n^c)$ be enumerating all possible random choices, where T is the time it takes to compute the generator. However, this might provide no speed-up.

In cryptography, we wanted the generator to be computable in polynomial time, and we wanted it to be secure against polynomial adversaries (recall that (s, ϵ) were super-polynomial). But for simulations of probabilistic algorithm we are not interested in these adversaries. The generator does not need to be good for all polynomials, but only for a fixed polynomial–that is, we are interested in the 2^n possible circuits resulting from hard-coding x.

Further, we simply need the running time of the generator to be no longer than exponential in length of input of generator: if $T(n) = 2^{l(t(n))}, t(n) > c \log n$ then the right answer can be found in $\leq 2^{O(t(n))}$.

Creating an efficient generator of this type (running in no longer than exponential time in length of input) is much easier to do than finding pseudo random generators in the cryptographic schemes we encountered before. Further, such generators are not equivalent to 1-way functions. Results relate to the worst-case bounds, and not to the average case bounds.

We now define an average case complexity for circuit complexity.

DEFINITION 2 *Let $f : \{0,1\}^l \to \{0,1\}$. The circuit complexity $CC(f) = \min\{S :$ there is a circuit C of size S such that C computes $f\}$.*

We define a notion hardness $H(f)$, as we defined hard-core predicates:

DEFINITION 3 *$H(f) = \min\{S :$ there is a circuit C of size $\leq S$ such that $\mathbf{Pr} f(x) = C(x) \geq 1/2 + 1/S\}$.*

We say that the hardness is $H(f) = h$ if given a circuit of size $< h$ can predict f on no more than a small fraction $+\ 1/2$ of the inputs: $\mathbf{Pr} x C(x) = f(x) \leq 1/2 + \frac{1}{h-1}$. Further, we could say f is (S, ϵ) hard if for every circuit of size $\leq S$, $\mathbf{Pr} x C(x) = f(x) \leq 1/2 + \epsilon$; then $h(f) = \max\{S : f \text{ is } (S, 1/S)\text{-hard}\}$.

When we discussed circuit complexity, we said that there exists $f : \{0,1\}^l \to \{0,1\}$ such that $CC(f) \geq 2^{l(1-o(1))}$, where CC is the circuit complexity and f is $\Omega(2^l/\epsilon)$. Using Chernoff bounds, it is not hard to show that hardness of f is $H(f) \geq 2^{l(1/3-o(n))}$, where $f : \{0,1\}^l \to \{0,1\}$.

CLAIM 1
For every $f : \{0,1\}^l \to \{0,1\}$ there is a circuit of size $O(2^{l/2})$ that computes f on a fraction $> 1/2 + 1/2^{l/2}$ of the inputs.

PROOF: On input x:

- Use the same $l/2$ bits as a subset
- Answer correctly on them.
- Guess most likely answer on the rest.

If $x = x : o^{l/2}$? then we output the right answer, and then output $f(x)$. Otherwise, we output the answer that is correct for majority of remaining $2^l - 2^{l/2}$ inputs. Then, the probability that the circuit computes f correctly is

$$\begin{aligned} \mathbf{Pr} x C(x) = 1 &\geq \frac{1}{2^{l/2}} + (1 - \frac{1}{2^{l/2}})\frac{1}{2} \\ &= 1/2 + \frac{1}{2^{l/2-1}} \end{aligned}$$

□

In fact, for every $f : \{0,1\}^l \to \{0,1\}$ there is a circuit C of size $2^{l(1/3+o(n))}$ that computes a function close to f: $\mathbf{Pr} x C(x) = f(x) \geq 1/2 + \frac{1}{2^{l(1/3+O(1))}}$. One example is that of a random hash function: $h : \{0,1\}^l \to \{0,1\}^{l/3}$. Let $y \in \{0,1\}^{l/3}$, and it stores the majority $\{f(x) : h(x) = y\}$. That is, for every string y, we look at the preimages, and determine is we see more often a 1 or a 0.

In the next four lectures, we will see the result of Nisan-Wigderson: (1988):

THEOREM 2
Suppose some language is solvable in deterministic time $L \in DTIME(2^{O(n)})$ and $\epsilon > 0$, such that for every large enough hn, $H(L_n) \geq 2^{\epsilon n}$, for ϵ bounded away from 0. Let L_n

be the restriction of L to inputs of length n: i.e. L_n is a machine that given an input of length n decides if x is in the language. Then there is a family of pseudo random generators $G_m : \{0,1\}^t(n) \to \{0,1\}^n$ computable in poly(m) time, with $t(m) = O(\log m)$ that are $(m^2, 1/m)$ pseudo random and further, $\mathbf{P} = \mathbf{BPP}$.

We will also see the result of Impagliazzo-Wigderson (1997):

THEOREM 3
Suppose there is $L \in DTIME(2^{O(n)})$ and $\epsilon > 0$, such that $CC(L_n) \geq 2^{\epsilon n}$ for all sufficiently large n. Then there is $L' \in DTIME(2^{O(n)}), \epsilon' > 0$, such that $H(L_n) > 2^{\epsilon' n}$ for all sufficiently large n and $\mathbf{P} = \mathbf{BPP}$.

So either circuits can get more than polynomial speed up over deterministic algorithms, or probabilistic algorithms do not given more than polynomial speed up over deterministic algorithms. This ties the average-case and worst-case complexity measures together:

Suppose there is $L \in DTIME(2^{o(n)})$, such that for all C and sufficiently large n, $CC(L_n \geq n^c$, then $\mathbf{BPP} = \bigcap DTIME(2^{n^\epsilon})$. This conjecture follows for certain restrictions of the previous two theorems. However, this type of argument uses a non-uniform reduction. Suppose for every problem there exists some hard problem, say in **BPP**, then the easy is in exptime. And, if exptime is hard, then **BPP** easy. So this is an interesting type of reduction. Ultimately, if the problem started from is hard, then the generator is hard to break. This is usually shown by the contrapositive: if we can break the generator, then we can solve the hard problem. Another result of Impagliazzo-Wigderson (1998) does the above in uniform way. If $\mathbf{BPP} \neq$ exp, then for every **BPP** problem L, and $\epsilon > 0$, there is an algorithm that runs in time 2^{n^ϵ} (it does not really solve, but solves on almost an inputs of length n. That is, for infinitely many n, $\mathbf{Pr} x A(x) = \text{rightanswer} > 1 - 1/n$.

Finally, there is a way of seeingAM as a version of **NP**. A pseudo random generator can give **NP** simulations of AM. This result by Arvin Kobler, Milterson, and ...

THEOREM 4
We define deterministic circuit complexity (N-CC) as the size of the smallest non-deterministic circuit for that function. Suppose there is $L \in NTIME(2^{O(n)})$ and $\epsilon > 0$ such that the deterministic circuit complexity $N - CC(L_n) \geq 2^{\epsilon n}$ for sufficiently large n, the $AM \subset$ **NP**.