COMS 6998: Unconditional Lower Bounds and Derandomization **Spring 2024**

Lecture 1: January 19, 2017

Lecturer: Rocco Servedio *Scribes: Ashvin Jagadeesan, Sam Wang*

1 Course Overview

Unconditional Lower Bounds and Derandomization is a course on advanced topics in complexity theory, the study of the inherent difficulties of computational problems. In the context of this class, we will ask the following question: *why are some functions hard to compute*?

We can divide complexity theory into two strands. The first is *high-level complexity*. This strand concerns itself with fundamental, somewhat philosophical questions around computation. These questions involve complexity classes **P**, **NP**, **BPP**, **PSPACE**, etc. These questions are typically deep and simple to ask, yet difficult to answer. Here are some examples:

- Are there natural problems that have **no** efficient algorithms? In other words, is $\mathbf{P} = \mathbf{NP}$?
- Is randomness useful for efficient computation? In other words, is $\mathbf{BPP} = \mathbf{P}$?

Questions involving high-level complexity are typically conditional results. That is, we prove theorems by assuming unresolved conjectures about complexity classes.

The second strand is *low-level complexity*. Because of the challenges in proving unconditional results for relatively unrestricted models such as polynomial-size circuits, low-level complexity often works with highly restricted models. Even though we limit our models, results in the second can inform insights regarding high-level complexity.

This course focuses on two major topics within the second strand: unconditional lower bounds in the direction of $\mathbf{P} \neq \mathbf{NP}$, and derandomization techniques in the direction of $\mathbf{P} = \mathbf{BPP}$. We'll look at different restricted models of computation, including

- Boolean formulae;
- Shallow circuits;

- DNF/CNF formulae;
- Decision trees;
- Branching programs;
- Polynomial models of computation;
- Restricted communication protocols.

With the models, we aim to establish the following kinds of results:

- Worst-case lower bounds;
- Average-case lower bounds;
- Pseudorandom generators;
- Deterministic approximate counting schemes.

Relevant readings:

- HH Survey Chapters 1 and 4.1 [HH23].

2 Example Models

2.1 Boolean Formulae

A Boolean formula is a function on binary inputs represented by a tree (typically a binary tree), where the leaves are Boolean literals (e.g. x_1, $\overline{x_3}$) and internal nodes are logical AND and OR gates. Notice that, WLOG, we can assume that all negations occur at the leaf level by applying De Morgan's laws. The size of a formula is the number of leaves and the depth is the depth of the tree.

2.2 DNF/CNF

DNFs and CNFs are depth-2 Boolean formulas. A DNF is an OR of ANDs and a CNF is an AND of ORs. The size of a DNF or CNF is the number of clauses and the width is the number of literals in any clause. Unlike with formulas, we typically assume unbounded fan-in, meaning that each gate permits an unlimited number of variables as inputs.

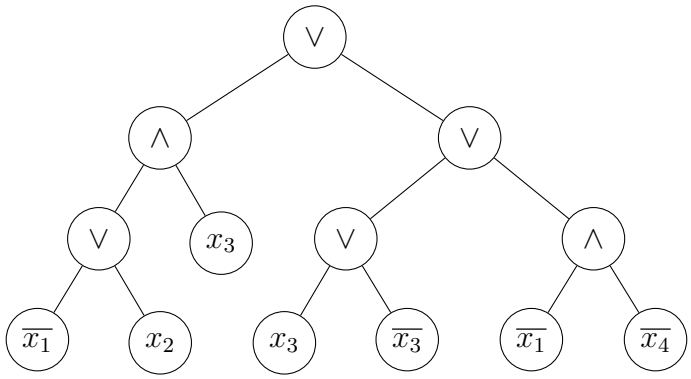

Figure 1: A tree representation of a Boolean formula.

2.3 Constant-Depth Circuits

The class $\mathbf{AC}^0_{d,s}$ comprises circuits of depth d and size s that involve only AND, OR, and NOT gates. Whereas a formula's underlying structure is a tree, a circuit is represented by a directed graph and allows, among other features, self-contained loops. Therefore, all formulas are circuits, but circuits also allow previous calculations to be "reused."

For instance, if n is a power of 2, then we can apply the following recursive relation to compute the parity function $PAR(x_1, \ldots, x_n)$:

$$\begin{aligned} PAR(x_1, \ldots, x_n) =& (PAR(x_1, \ldots, x_{n/2}) \wedge \overline{PAR(x_{n/2+1}, \ldots, x_n)}) \vee \\ & (\overline{PAR(x_1, \ldots, x_{n/2})} \wedge PAR(x_{n/2+1}, \ldots, x_n)). \end{aligned}$$

In a circuit, we can reuse our computations for the parity of each half, whereas in a circuit, we have to compute the parity of each half twice. Constant-depth circuits are simply circuits with a fixed value for d.

2.4 Polynomials over $\mathbb{F}_2$

We consider polynomials in $\mathbb{F}_2^n \to \mathbb{F}_2$, where all operations are performed modulo 2. The sparsity of a polynomial is the number of monomials and the degree is the maximum degree of any monomial. The class DEG_d comprises all boolean functions $f : \{0,1\}^n \to \{0,1\}$ computable by degree$-d$ $\mathbb{F}_2$ polynomials.

3 Computational Hardness

One of our big goals will be to establish hardness results. The dream result would be to prove that $\mathbf{P} \neq \mathbf{NP}$. Unfortunately, there is very little progress towards proving this. How might we try to prove this claim? A natural way would be to prove that there are boolean functions whose corresponding language is in **NP** that have super-polynomial lower bounds on circuits.

If we could prove that any family $(C_n)_{n\geq 1}$ of circuits solving any **NP**-complete problem must have size $\geq n^{\omega(1)}$, then we could get that $\mathbf{P} \neq \mathbf{NP}$ as an easy consequence. However, we don't know how to prove this. As a compromise, we will not consider *unrestricted circuits* but rather limited models of computation (primarily the ones discussed in the previous section). This compromise leads us to our definition of **worst case hardness**.

Definition 1 (Worst-Case Hardness)**.** *Let $\mathcal{C}$ be a class of n-variable functions $f : \{0,1\}^n \to \{0,1\}$. We say that $h : \{0,1\}^n \to \{0,1\}$ is worst-case hard for $\mathcal{C}$ if $h \notin \mathcal{C}$.*

Notice that any instance of a wrong result over the possible inputs would disqualify a function h from $\mathcal{C}$, so there could be a function that is easily solvable on every input except for one intractable case. To capture the computational hardness in the broader case, we introduce *average-case hardness* that captures the fraction of inputs correctly computed.

Definition 2 (Correlation of Boolean functions)**.** *Let $f, h : \{0,1\}^n \to \{0,1\}$ be boolean functions. Let $\mathcal{D}$ be some distribution over $\{0,1\}^n$. The correlation between f and h over distribution $\mathcal{D}$ is given by*

$$\mathrm{Cor}_{\mathcal{D}}[f,h] = |\mathrm{Pr}_{x\sim\mathcal{D}}[f(x) = h(x)] - \mathrm{Pr}_{x\sim\mathcal{D}}[f(x) \neq h(x)]|.$$

Correlation can be viewed directly in terms of the error rate:

$$\mathrm{Cor}_{\mathcal{D}}[f,h] = 2\left|\frac{1}{2} - \mathrm{Pr}_{x\sim\mathcal{D}}[f(x) \neq h(x)]\right|.$$

If $f, h : \{0,1\} \to \{-1,1\}$, then

$$\mathrm{Cor}_{\mathcal{D}}[f,h] = \left|\mathbb{E}_{x\sim\mathcal{D}}[f(x)\cdot h(x)]\right|.$$

By inspection, $\mathrm{Cor}_{\mathcal{D}}[f,h]$ always takes on values in $[0,1]$, with 0 meaning no correlation and 1 meaning maximal correlation. Notice that $\mathrm{Cor}_{\mathcal{D}}[f,h] = 1$ if $f = h$ or $f = \overline{h}$.

Typically, we let $\mathcal{D}$ be the uniform distribution, but this definitional flexibility allows for alternate distributions over inputs.

We can extend this definition of correlation to classes of Boolean functions. Specifically, let $\mathcal{C}$ be a class of functions and let h be a function, then

$$\mathrm{Cor}_{\mathcal{D}}[\mathcal{C}, h] = \max_{f \in \mathcal{C}} \mathrm{Cor}_{\mathcal{D}}[f, h].$$

Definition 3 (Average-case hardness)**.** *Let $\mathcal{C}$ be a class of functions $f : \{0,1\}^n \to \{0,1\}$. Let $\mathcal{D}$ be distribution over $\{0,1\}^n$ and $\epsilon > 0$. We say that $h : \{0,1\}^n \to \{0,1\}$ is ϵ-hard for $\mathcal{C}$ with respect to $\mathcal{D}$ if*

$$\left| Pr_{x \sim \mathcal{D}}[f(x) = h(x)] - \frac{1}{2} \right| \leq \epsilon$$

for all $f \in \mathcal{C}$.

Alternatively, h is ϵ-hard for $\mathcal{C}$ with respect to $\mathcal{D}$ if $\mathrm{Cor}_{\mathcal{D}}[\mathcal{C}, h] \leq 2\epsilon$.

4 Randomized Algorithms

Randomization is a highly powerful paradigm in algorithms. Many of the most efficient known algorithms for problems involve randomization, and these methods tend to also be the simplest. To illustrate, here are two problems with highly efficient and simple randomized solutions.

4.1 Graph Reachability

Given a graph $G = (V, E)$, and nodes $s, t \in V$, the goal is to decide whether t is reachable from s by traversing edges in G. A randomized solution is to start at the node s and walk randomly, choosing a random neighbor as the next step, for $100n^2$ steps. It can be shown that if t is reachable, then we will have encountered it during the walk with at least 99% probability. Otherwise, we will never encounter t during the walk. This algorithm runs in $O(n^2)$ time, comparable to deterministic approaches like BFS and DFS, but uses only $O(\log n)$ space.

4.2 Polynomial Identity Testing

In this problem, the goal is to determine whether two given polynomials $p, q : \mathbb{R}^n \to \mathbb{R}$ are equivalent; namely, if $p(x) = q(x)$ for all $x \in \mathbb{R}^n$. Let $m = \deg(p) + \deg(q)$. A

simple randomized solution is to choose a test point x uniformly from $\{1, 2, \ldots, 3m\}^n$ and check if $p(x) = q(x)$. If $p \not\equiv q$, then by the Schwarz-Zippel lemma (not proved here),

$$\Pr[p(x) = q(x)] \leq \frac{\deg(p-q)}{3m} \leq \frac{1}{3}.$$

Repeating this procedure a constant number of times results in an $\exp(-\Omega(k))$ failure probability.

5 Derandomization

5.1 Psuedorandom Generators: The High-Level Idea

When making random choices, it is useful to have an unlimited supply of random coin tosses. However, much like time and space, we think of access to unbiased, independent coin tosses as a valuable resource. As a result, we would like to minimize the number of truly random coin tosses we use. A pseudorandom generator, or PRG for short, is a mathematical object that utilizes a small amount of randomness—in the form of a small sequence of truly random coin tosses—to generate a long sequence of coin tosses that appears random. Due to the deterministic nature of PRGs, this longer sequence is not actually random from an information-theoretic standpoint. As a result, we utilize PRGs not for their information-theoretic properties, but their computational properties.

The goal is to connect PRGs to concrete computational models, where the limited randomness of the PRG is sufficient to "fool" the model. We can approach this problem either by starting with a model and devising a PRG, or searching for models that an already instantiated PRG can fool.

5.2 Some Formal Definitions

To formally proceed with our goal, it is crucial to introduce concrete definitions before we move forward. Thus, we are first required to formalize the notion of *fooling*. Let $\mathcal{U}_n$ denote the uniform random variable over $\{0,1\}^n$.

Definition 4 (Fooling)**.** *Suppose that $f : \{0,1\}^n \to \{0,1\}$ is a boolean function, X is a probability distribution over $\{0,1\}^n$, and $\epsilon > 0$. We say that X fools f with error ϵ, or ϵ-fools f if*

$$|\Pr_{x \sim X}[f(x) = 1] - \Pr_{x \sim \mathcal{U}_n}[f(x) = 1]| \leq \epsilon.$$

More generally, we can consider a real-valued function $f : \{0,1\}^n \to \mathbb{R}$. We say that X fools f with error ϵ if:

$$|\mathbb{E}[f(X)] - \mathbb{E}[f(\mathcal{U}_n)]| \leq \epsilon.$$

If $\epsilon = 0$, then we say that X *perfectly fools* f.

The definition above underscores our previous discussion, that is, even if X is not the uniform distribution, it appears indistinguishable from the uniform distribution from f's perspective. This indistinguishability typically, from f's perspective, arises from f's limited computational resources. Conversly, if X *does not ϵ-fool* f, then f is called a *distinguisher* for X.

With the notion of *fooling* formalized, we can now move on to formalizing the definition of a PRG.

Definition 5 (PRGs)**.** *Suppose that $f : \{0,1\}^n \to \mathbb{R}$ and $G : \{0,1\}^s \to \{0,1\}^n$ are functions and $\epsilon > 0$. The parameter s is called the seed-length. We say that G is an ϵ-PRG for f if $G(\mathcal{U}_s)$ fools f with error ϵ. That is,*

$$|\mathbb{E}[f(G(\mathcal{U}_s))] - \mathbb{E}[f(\mathcal{U}_n)]| \leq \epsilon.$$

In this case, we also say that G fools f with error ϵ.

Despite the promising claims of PRGs, there is one unfortunate fact: for any non-trivial PRG, there exists a function that is not fooled by the PRG. (Note: This theorem was not covered in class.)

Theorem 6 (Impossibility of fooling all functions)**.** *Let $G : \{0,1\}^s \to \{0,1\}^n$ where $s < n$. Then there exists some function $f : \{0,1\}^n \to \{0,1\}$ such that G does not 0.49-fool f*

Proof. Let f be a boolean function such that it outputs 1 in the image of G. When drawing $x \sim \{0,1\}^n$, the probability that it is in the image of G is $2^s/2^n$. This is due to the fact that there are 2^s strings in the image of G and 2^n strings in the set $\{0,1\}^n$. Based on this construction, we have that

$$|Pr_{x\sim G(\mathcal{U}_s)}[f(x) = 1] - Pr_{x\sim \mathcal{U}_n}[f(x) = 1]| = \left|1 - \frac{2^s}{2^n}\right|.$$

Now, due to the fact that $s < n$, it is true that if $\epsilon = 0.49$

$$\left|1 - \frac{2^s}{2^n}\right| > \epsilon$$

Thus, G does not 0.49-fool f. ■

In light of this theorem, the best we can hope for is to fool a large subset of functions.

We can similarly extend this definition to classes of functions. Intuitively, we want to create a random variable X without using n degrees of randomness such that results using X are comparable to results involving the truly random variable $\mathcal{U}_n$. A pseudo-random generator uses only s bits of true randomness from a *seed* to simulate n bits of randomness. The goal is to find an ϵ-PRG with $s \ll n$ so that we can loop through all values in $\{0,1\}^s$ and simulate the results of a randomized algorithm.

To that end, if $\mathcal{C}$ is a class of functions $f : \{0,1\}^n \to \mathbb{R}$, we say that G is an ϵ-PRG for $\mathcal{C}$ if G ϵ-fools f for all $f \in \mathcal{C}$.

Lemma 7 (Nonexplicit PRGs are easy)**.** *For any class of functions $\mathcal{C}$, there exists an ϵ-PRG for $\mathcal{C}$ with $s = \log\log|\mathcal{C}| + 2\log\frac{1}{\epsilon} + O(1)$.*

The high level proof idea revolves around utilizing the *probabilistic method*. That is, we will show that such a PRG exists without an explicit construction.

Proof. Let G be a uniform random mapping from $\{0,1\}^s$ to $\{0,1\}^n$. In other words, for every $y \in \{0,1\}^s$, we let $G(y) = 0$ or $G(y) = 1$ with equal probability. Notice that for each fixed s-bit number y, $f(G(y))$ is a random 0/1 number with

$$\mathbb{E}_G[f(G(y))] = \mathbb{E}_{\mathcal{U}_n}[f(\mathcal{U}_n)].$$

Applying Chernoff bounds and expanding out the definition of expectation, we get

$$\Pr_G\left[\left|\mathbb{E}_{\mathcal{U}_n}[f(\mathcal{U}_n)] - \frac{1}{2^s}\sum_{y\in\{0,1\}^s} f(G(y))\right| \geq \epsilon\right] \leq 2e^{-2\epsilon^2\cdot 2^s}.$$

Then, we union bound over all functions in $\mathcal{C}$ to obtain the probability of ϵ-fooling $\mathcal{C}$:

$$\Pr[G \text{ doesn't fool } \mathcal{C}] \leq 2|\mathcal{C}| \cdot e^{-2\epsilon^2\cdot 2^s},$$

for which $s = \log\log|\mathcal{C}| + 2\log\frac{1}{\epsilon} + O(1)$ suffices. ■

Observe that this construction produces highly efficient PRGs. For instance, even the class of all poly(n)-size circuits, with $2^{\text{poly}(n)}$ members, only requires $s = O(\log\frac{n}{\epsilon})$ many bits of randomness. Unfortunately, this method requires randomness in order to produce G. If our goal is to derandomize a randomized algorithm, then we seek *explicit* constructions of PRGs which do not involve any randomness in the construction process.

5.3 Deterministic Algorithms from PRGs

It may appear that pseudorandom generators only kick the can down the road, as it requires a random seed of length s to generate a seemingly random number of length n. But we can resolve this problem by looping through all 2^s seeds. As a consequence, a sufficiently efficient PRG can give rise to efficient deterministic algorithms for computational problems.

Lemma 8. *Suppose $G : \{0,1\}^s \to \{0,1\}^n$ is an explicit PRG that ϵ-fools a class $\mathcal{C}$ of functions $f : \{0,1\}^n \to \mathbb{R}$. Then, there exists a deterministic algorithm running in $2^s \cdot \text{poly}(n)$ time that outputs a value in*

$$\left[\mathbb{E}_{\mathcal{U}_n}[f(\mathcal{U}_n)] - \epsilon, \mathbb{E}_{\mathcal{U}_n}[f(\mathcal{U}_n)] + \epsilon\right].$$

Proof. For each $y \in \{0,1\}^s$, compute $f(G(y))$, and output the average. There are 2^s such points and each computation can be completed in polynomial time, so the overall time is $2^s \cdot \text{poly}(n)$. ■

We can view any polynomial-time randomized algorithm on some input x as a polynomial-size circuit C_x for x that accepts as input a sequence of random coin tosses $r = (r_1, \ldots)$. Therefore, we can think of fooling a class of circuits C_x where x is every possible deterministic input. Combined with Lemma 8, the next result follows naturally:

Corollary 9. *If there exists an explicit PRG G for the class $\mathcal{C}$ of all $\text{poly}(n)$-size circuits with a seed length such that $s = \text{poly}(n)$, then $\mathbf{P} = \mathbf{BPP}$.*

5.4 Worst-Case Lower Bounds from PRGs

Finally, we present a connection between computational hardness and derandomization.

Lemma 10 (PRG $\implies$ worst-case lower bounds)**.** *Let $\mathcal{C}$ be a class of functions $f : \{0,1\}^n \to \{0,1\}$ that is closed under restrictions (i.e. fixing any of the inputs as either 0 or 1 will produce a function that is still in $\mathcal{C}$). Let $G : \{0,1\}^s \to \{0,1\}^n$ be an explicit ϵ-PRG for $\mathcal{C}$ where $s < n$ and $\epsilon < 1/2$. Define $h : \{0,1\}^{s+1} \to \{0,1\}$ as*

$$h(x) = \begin{cases} 1 & \text{some string in the range of } G \text{ starts with } x \\ 0 & \text{otherwise} \end{cases}.$$

Let $\mathcal{C}'$ be all restrictions of functions in $\mathcal{C}$ by fixing the last $n-s-1$ bits, leaving the first $s+1$ still alive. Then, $h \notin \mathcal{C}'$.

Proof. For any string $t \in \{0,1\}^s$, $h(G(t)_{1,\dots,s+1}) = 1$ since clearly the first $s+1$ bits of $G(t)$ are a prefix for $G(t)$, so

$$\mathbb{E}[h(G(\mathcal{U}_s)_{1,\dots,s+1}] = 1.$$

At the same time,

$$\mathbb{E}[h(\mathcal{U}_{s+1})] \leq \frac{1}{2}$$

since the image of G has at most 2^s elements. Therefore, $h' \notin \mathcal{C}'$. ■

References

[HH23] Pooya Hatami and William Hoza. Theory of unconditional pseudorandom generators. *Electronic Colloquium on Computational Complexity*, Apr 2023. https://eccc.weizmann.ac.il/report/2023/019/. 1