Lecture 01: Probabilistic Complexity Classes, Error Reduction, and Average Case Complexity

Topics In Complexity and Pseudorandomness (Spring 2017)
Rutgers University
Swastik Kopparty
Scribes: Ross Berkowitz & Ana Uribe

1 Introduction

P, a main object of study in complexity, is the class of all languages that can be recognized by a polynomial time Turing Machine. Formally

Definition 1. *A Language $L \subset \{0,1\}^*$ is said to be in **P** if there is some c and a Turing Machine (algorithm) A, which runs in time $O(n^c)$ such that for all x $A(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{else} \end{cases}$*

NP is the class of all languages such that membership in L can be certified by a polynomial time Turing Machine.

Definition 2. *A Language $L \subset \{0,1\}^*$ is said to be in **NP** if there is some c, and an algorithm $A(x,w)$ that runs in time $O(|X|^c)$ such that for all x*

- *If $x \in L$ then $\exists w$ such that $A(x,w) = 1$*
- *If $x \notin L$ then $\forall w$ $A(x,w) = 0$*

For a typical example of an **NP** problem, let L be the set of all 3-colorable graphs (it is a big open question if L is in **P**). Consider the algorithm $A(x,w)$ which takes as input a graph x and a coloring w and outputs 1 if w is a valid 3-coloring and 0 if it is not. It is not hard to construct a polytime algorithm A that verifies proper colorings, so this shows that {3-colorable graphs} $\in$**NP**. For contrast {2-colorable graphs} $\in$**P**. To summarize: **P** means efficiently recognizable and **NP** means efficiently checkable.

In this class we will be interested in randomized polynomial time, and will set up some complexity classes dealing with randomness.

Definition 3 (**RP** Randomized Polytime)**.** *$L \in$**RP** if there exists an algorithm $A(x,r)$, where r is a length m string chosen uniformly at random (where m is a function of $|x|$), such that*

- *A runs in time $O(|x|^c)$*
- *if $x \in L$ then $\Pr_{r \in \{0,1,\}^m}\left[A(x,r) = 1\right] \geq \frac{1}{2}$*
- *if $x \notin L$ then $\forall r \in \{0,1\}^m$, $A(x,r) = 0$*

A few important points to note:

- The algorithm A is deterministic, the randomness is in the randomized input r.
- **RP** has one sided error- if A outputs 1, then x is certainly in L.
- The guarantee is for *all* possible inputs x. This is separate from average case complexity which might say something for most x. Here our quantifier is for most randomized strings r.

A few more observations

- $\mathbf{P} \subset \mathbf{RP} \subset \mathbf{NP}$
- People believe that $\mathbf{P}=\mathbf{RP} \neq \mathbf{NP}$
- Random vs. Deterministic really only matters in polynomially bounded time, else we could enumerate over all sources of randomness.
- We can replace the success probability of $\frac{1}{2}$ with any fixed number in $(0, 1)$ and not change the class. If we use the constant 1, then we just have **P**.

Definition 4 (**BPP** Bounded error Probabilistic Polytime)**.** *$L \in$ **BPP** if $\exists A(x,r)$ with $O(|x|^c)$ runtime such that*

- *if $x \in L$ then* $\Pr_{r \in \{0,1\}^m}[A(x,r) = 1] \geq 0.9$
- *if $x \notin L$ then* $\Pr_{r \in \{0,1\}^m}[A(x,r) = 0] \geq 0.9$

Note that the 0.9 in this definition can be replaced with any constant bigger than $\frac{1}{2}$ and the class will stay the same.

Next we will talk about circuits. A circuit depends on the input size (the number of wires used is a function of the number of bits input), and because of this they can do undecidable things (see the subsequent bullets for an example). A circuit sequence (which will sometimes just be called a circuit) is a sequence $(C_n)_{n=1}^{\infty}$ such that C_n takes inputs of size n.

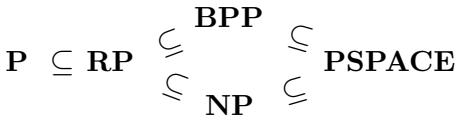
Definition 5 (**P/poly**)**.** *A Language L is in **P/poly** if $\exists$ a circuit sequence such that*

- *The number of gates in C_n is $O(n^c)$*
- *if $x \in L$ then $C_{|x|}(x) = 1$*
- *if $x \notin L$ then $C_{|x|}(x) = 0$*

Things we know:

- $\mathbf{P} \subset \mathbf{P/poly}$. It is possible to encode the execution of an algorithm as wires tracking how the memory evolves.

- $\mathbf{P/poly} \not\subset \mathbf{P}$. For example consider the language $L = \{0^n \mid n \text{ encodes a non-halting Turing Machine}\}$. L isn't even decidable, and certainly not in $\mathbf{P}$, but as a circuit it is doable (with trivial circuits, because the circuit builder picks his circuit for each n and so has unreasonable power).



The Story So Far

We have reason to believe $\mathbf{P} = \mathbf{BPP}$ and $\mathbf{P} \neq \mathbf{NP}$.

Theorem 6. *If* $\boldsymbol{P = NP}$ *then* $\boldsymbol{P = BPP}$.

We will prove this theorem later in the course. Note, as of the moment $\mathbf{BPP} \not\subset \mathbf{NP}$, so this is interesting. One of the big goals of this class will be to understand what extra power randomness gives us.

2 Error Reduction

2.1 Error Reduction for RP

Suppose we have $A(x, r)$ such that

- if $x \in L$ then $\Pr_{r \in \{0,1,\}^m} \left[A(x, r) = 1\right] \geq 0.5$
- if $x \notin L$ then $\forall r \in \{0, 1\}^m$, $A(x, r) = 0$

We want to reduce our error. We make the new algorithm $A'(x, r_1, r_2, \ldots, r_t)$ where $r_i \in \{0, 1\}^m$ by running A on each of the randomized inputs r_i independently. Formally

$$A'(x, r_1, \ldots, r_t) = \begin{cases} 1 & \text{if } A(x, r_i) = 1 \text{ for some } i \\ 0 & \text{if } A(x, r_i) = 0 \text{ for all } i \end{cases}$$

Note that if $x \notin L$ then we again have that A' outputs 0. However if $x \in L$ then the probability that A' misses is $1 - \Pr_{r_i}[A(x, r_i) = 0] \geq 1 - (\frac{1}{2})^t$. So we can in this way guarantee error $1 - \epsilon$ using $\log(\epsilon)$ many samples, or an exponentially small error by choosing $t = O(n)$.

2.2 Error Reduction for BPP

Recall that in the setup of **BPP** we have $A(x, r)$ such that

- if $x \in L$ then $\Pr_{r \in \{0,1\}^m}[A(x, r) = 1] \geq 0.9$

- if $x \notin L$ then $\Pr_{r \in \{0,1\}^m}[A(x,r) = 0] \geq 0.9$

We want to reduce error. A pretty good strategy is to take t inputs $r_1, \ldots, r_t$ at random and run $A(x, r_i)$ for each random string r_i and then output $Maj(A(x, r_i), 1 \leq i \leq t)$ The question now is to analyze the accuracy of this algorithm. Note that $\{A(x, r_i)\}_{i=1}^t$ is a set of independent identically distributed random variables. Without loss of generality say that $x \in L$. In this case we know that

$$\mathbb{E}_{r_i \in \{0,1\}^m}\left[\sum_{i=1}^{t} A(x, r_i)\right] = \sum_{i=1}^{t} \mathbb{E}[A(x, r_i)] = .9t$$

Theorem 7 (Chernoff Bound)**.** *Let $X_1, \ldots, X_t$ independent random variables supported on $[-1, 1]$ with means $\mathbb{E}[X_i] = \mu_i$. Then*

$$\Pr\left[\left|\frac{\sum X_i}{t} - \frac{\sum \mu_i}{t}\right| > \epsilon\right] \leq e^{-\frac{\epsilon^2 t}{10}}$$

The point is that this probability is exponentially small in t whenever ϵ is constant. Chernoff applies exactly in our setting above, and immediately implies that

$$\Pr[Maj(A(x, r_i)) \neq 1] = \Pr\left[\frac{1}{t}\sum X_i \leq 0.5\right] \leq \Pr\left[\left|\frac{1}{t}\left(\sum x_i - \sum \mu_i\right)\right| > 0.4\right] \leq e^{-\frac{0.16t}{10}}$$

If $x \notin L$ the same argument shows that $Maj(\{A(x, r_i)\}) = 0$ with probability $\geq 1 - e^{-\frac{0.16t}{10}}$. So we have error probability e^{-ct} for some constant c, and t a parameter which we get to choose. Increasing t will improve the accuracy of A' while we use mt random bits. For t chosen to be a large constant we can get correct output with probability 0.999. When t is taken to be n we get error probability exponentially small in n.

Theorem 8 (Aldeman)**.** ***BPP*** $\subset$ ***P/poly***

A bit of philosophy: This result suggests that **P** = **BPP** . Why should **P/poly** be better than **P** ? How does it help to use a different algorithm on different n?

Proof. Take $L \in$ **BPP** . We have $A(x, r)$ a **BPP** algorithm for L. Suppose you had a randomized algorithm with the property that for most r, $A(x, r)$ is the right answer. We want to make A deterministic. For every x there is a large set of $r \in \{0,1\}^m$ which works for that specific x. If we could find a single r^* that works for all $x \in L$, then we could hard wire r^* into the circuit, and have an algorithm which is always correct (note that r^* depends on n, and so while this is ok for **P/poly** , it would cause trouble for an algorithm in **P** . Such existence results work for **P/poly** as the circuit designer is unbounded, but an algorithm for **P** would have trouble finding r^*).

Step 1: Error reduce our **BPP** algorithm to get A' with failure probability $\exp(-\Omega(n^2))$ (using $t = n^2$ in the above error reduction).

Step 2: It follows immediately that

$$\Pr_{\substack{x \in \{0,1\}^n \\ r \in \{0,1\}^m}}[A(x, r) \text{ fails}] < e^{-\Omega(n^2)}$$

Step 3: Therefore there exists some r^* such that

$$\Pr_{x\in\{0,1\}^n}[A(x,r) \text{ fails}] < e^{-\Omega(n^2)}$$

We now claim that this r^* works for all inputs x, that is $A'(x,r^*)$ is a deterministic polynomial time algorithm (depending on n) which checks membership in L. Indeed, if for even a single $x \in \{0,1\}^n$ we had that $A'(x,r^*)$ failed, then it would be the case that r^* failed with probability at least 2^{-n}, contradicting our conclusion in step 3. Now the circuit designer can simply build for each n, a circuit running $A'(x,r^*)$, showing that L is in **P/poly** . □

Note that this proof required using a lot of randomness we used n^2m bits of randomness where m was the amount of randomness used initially in the **BPP** algorithm A. So we used far more bits of randomness than the length of our input and spread our errors out among the randomness. In a later class we will see how to reduce the amount of randomness we used.

3 Average Case Complexity

Given a function $f : \{0,1\}^n \to \{0,1\}$ and algorithm A

$$\Pr_{x\in\{0,1\}^n}[A(x) = f(x)] \geq 0.99$$

Definition 9. *We say $f : \{0,1\}^n \to \{0,1\}$ is δ-hard for circuits of size s if for all circuits C of size $\leq s$*

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

Note: For all s, no function is $\frac{1}{2}$-hard because the circuit that outputs all 1 or all 0 (depending on $\text{Maj}(f(x) : \forall x \in \{0,1\}^n)$) does not satisfy the above statement.

Definition 10. *We say $f : \{0,1\}^n \to \{0,1\}$ is hard for circuits of size s if it is 0-hard. In other words for all circuits C of size $\leq s$*

$$\exists x \text{ s.t. } C(x) \neq f(x)$$

Theorem 11. *$\exists f$ which is 0-hard for circuits of size $o(\frac{2n}{n})$*

Proof. Let $f : \{0,1\}^n \to \{0,1\}$ be a uniform random function. Fix a circuit C of size s then

$$\Pr_f[C(x) = f(x) \forall x] = \frac{1}{2^{2^n}}$$

(because there are 2^{2^n} boolean functions on $\{0,1\}^n$)

This implies that

$$\Pr_f[\exists C \text{ of size } \leq s \text{ s.t. } C(x) = f(x) \forall x] = \frac{\text{num. of circuits of size} s}{2^{2^n}} \overset{\star}{\leq} \frac{s^{O(s)}}{2^{2^n}} = 2^{O(s \log s) - 2^n}$$

This last value is less than 0.001 if $s < o(\frac{2^n}{n})$ so

$$\Pr_f[f \text{ is hard for size } s] \geq 0.999$$

$\star$ Why are there $s^{O(s)}$ circuits of size s?
Pick 2 inputs for each gate from the n inputs or any of the outputs of the other gates $\rightarrow \leq (s+n)^{2s}$ choices
Pick the type of gate $\rightarrow \leq 5^s$ choices
This yields at most $(s+n)^{2s}5^s = O(s)^O(s) = s^{O(s)}$ different circuits of size s. □

Note: We don't know the 0-hard function explicitly, we just know it exists.

Theorem 12. *For all $\epsilon > 0$ there exists f which is $(\frac{1}{2} - \epsilon)$-hard for circuits of size $o(\frac{2^n}{n})$*

Proof. Pick f at random.

Fix C for size $s > n$, using the Chernoff bound (with $\mu = \frac{1}{2}$) we get

$$\Pr[C(x) = f(x) \text{ for } \geq (\frac{1}{2} + \epsilon) \text{ fraction of the } x\text{'s}] \leq e^{\epsilon^2 \frac{2^n}{10}}$$

$$\Pr[\exists C \text{ of size } s \text{ such that } C(x) = f(x) \text{ for } \geq (\frac{1}{2}+\epsilon) \text{ fraction of the } x\text{'s}] \leq s^{O(s)} e^{\epsilon^2 \frac{2^n}{10}} \leq 2^{O(s \log s) - \epsilon^2 \frac{2^n}{10}}$$

So for $s = o\left(\frac{2^n \epsilon^2}{n + 2 \log 1/\epsilon}\right)$ this quantity is $<< o(1)$ and

$$\Pr[fis(\frac{1}{2} - \epsilon)\text{-hard}] = 1 - o(1)$$

□

Theorem 13. ***Hard Core Lemma (Impagliazzo 95)*** *Suppose $f : \{0,1\}^n \rightarrow \{0,1\}$ is δ-hard for circuits of size s then for all ϵ there exists some $H \subseteq \{0,1\}^n$, "the hard core" such that $|H| = 2\delta 2^n$ and for all circuits C of size $\leq \text{poly}(\epsilon, \delta)s$*

$$\Pr_{x \in H}[C(x) = f(x)] \leq \frac{1}{2} + \epsilon$$

Fact: If f is such that $\exists H$, $|H| = 2\delta 2^n$ and for all C of size s

$$\Pr_{x \in H}[C(x) = f(x)] \leq \frac{1}{2} + \epsilon$$

then f is $\delta - 2\delta\epsilon$ hard.
This is because is makes at least $2\delta(\frac{1}{2} - \epsilon)2^n$ errors on H so it makes at least $(\delta - 2\delta\epsilon)2^n$ errors on $\{0,1\}^n$

The strategy for the proof of the hard core lemma is

Suppose that for all H of size $2\delta 2^n$ there exists a C of size $\text{poly}(\epsilon, \delta)s$ such that

$$\Pr_{x \in H}[C(x) = f(x)] > \frac{1}{2} + \epsilon$$

and we want to show that there exists a circuit C^* of size s such that

$$\Pr_{x \in \{0,1\}^n}[C^*(x) = f(x)] \geq 1 - \delta$$

In order to do this, we'll need to use the minimax theorem which says the following

Theorem 14. ***Minimax theorem*** *We have an $n \times m$ matrix M of real number and two players R and C that simultaneously select a row i and a column j of M respectively.*

In this game, player R wants to maximize M_{ij} and player C wants to minimize M_{ij}.

If we have that R selects a probability distribution P over the rows and C selects a probability distribution Q over the columns then the expected output is

$$E[\text{output}] = \sum_{i=1}^{n} \sum_{j=1}^{m} P_i Q_j M_{ij}$$

The minimax theorem states that there exists a value v such that, in expectation, R can guarantee a value of at least v and C can guarantee a value of at most v.

We'll prove both the minimax theorem and the hard core lemma next lecture.