Lecture 8: Randomized Complexity Classes and Polynomials

Anup Rao

February 25, 2022

Probability Review

We start by reviewing a couple of useful facts from probability theory.

Lemma 1 (Markov's inequality). *If X is a non-negative random variable, then $\Pr[X > \ell \cdot \mathbb{E}[X]] < 1/\ell$.*

Proof

$$
\begin{aligned}
\mathbb{E}[X] &= \sum_k k \cdot \Pr[X = k] \\
&\geq \sum_{k > \ell} (\ell\, \mathbb{E}[X]) \cdot \Pr[X = k] \\
&= \ell\, \mathbb{E}[X] \cdot \sum_{k > \ell} \Pr[X = k],
\end{aligned}
$$

proving that $\Pr[X > \ell] \leq 1/\ell$. ∎

We shall need to appeal to the Chernoff-Hoeffding Bound:

Theorem 2. *Let $X_1, \ldots, X_n$ be independent random variables such that each X_i is a bit that is equal to 1 with probability $\leq p$. Then $\Pr[\sum_{i=1}^{n} X_i \geq pn(1 + \epsilon)] \leq 2^{-\epsilon^2 np/4}$.*

Finally, we need the following trick. Suppose we toss a coin which has a probability p of giving heads and $1 - p$ of giving tails. Let H denote the number of coin tosses before we see heads. Then

Fact 3. $\mathbb{E}[T] = 1/p$.

Proof

$$
\begin{aligned}
\mathbb{E}[T] &= p \cdot 1 + (1 - p) \cdot (\mathbb{E}[T] + 1) \\
\Rightarrow \mathbb{E}[T] &= 1 + (1 - p) \cdot \mathbb{E}[T] \\
\Rightarrow \mathbb{E}[T]\, p &= 1 \\
\Rightarrow \mathbb{E}[T] &= 1/p.
\end{aligned}
$$

∎

Randomized Classes

There are several different ways to define complexity classes involving randomness. A turing machine with access to randomness is just like a normal turing machine, except it is allowed to toss a random coin in each step, and read the value of the coin that was tossed.

BPP

We say that the randomized machine computes the function f if for every input x, $\Pr_r[M(x,r) = f(x)] \geq 2/3$, where the probability is taken over the random coin tosses of the machine M. **BPP** is the set of functions that are computable by polynomial time randomized turing machines in the above sense.

RP

We shall say that $f \in$ **RP** if there is a randomized machine that always compute the correct value when $f(x) = 0$, and computes the correct value with probability at least $2/3$ when $f(x) = 1$.

ZPP

Finally, we define the class **ZPP** to be the set of boolean functions that have an algorithm that *never* makes an error, but whose *expected* running time is polynomial in n.

Error reduction

The choice of the constant $2/3$ in these definitions is not crucial, as the following theorem shows:

Theorem 4 (Error Reduction in **BPP**). *Suppose there is a randomized polynomial time machine M, a boolean function f and a constant c such that $\Pr_r[M(x,r) = f(x)] \geq 1/2 + n^{-c}$. There for every constant d, there is a randomized polynomial time machine M' such that $\Pr_r[M'(x,r) = f(x)] \geq 1 - 2^{-n^d}$.*

Proof of Theorem 4:   On input x, the algorithm M' will run M repeatedly n^k times for some constant k (that we shall fix soon), and then output the majority of the answers. Let X_i the binary random variable that takes the value 1 only if the output of the i'th run is incorrect.

We have that $X_1, \ldots, X_{n^k}$ are independent random variables, and

each is equal to 1 with probability at most $1/2 - n^{-c}$. Thus,

$$\Pr[\sum_i X_i > n^k/2] = \Pr[\sum_i X_i > n^k(1/2 - n^{-c})(1/2)/(1/2 - n^c)]$$
$$\leq \Pr[\sum_i X_i > n^k(1/2 - n^{-c})(1 + 2n^{-c})]$$
$$< 2^{-O(n^{-2c})n^k/8}$$

Set k to be large enough so that this probability is less than 2^{-n^d}. ∎

By brute force search, we can easily prove:

Theorem 5. BPP $\subseteq$ EXP.

Since **RP** is the same as the set of functions for which a random witness is a good witness,

Theorem 6. RP $\subseteq$ NP.

Theorem 7. *Every function in* **BPP** *has polynomial sized circuits.*

The above theorem again easily following from the Chernoff-Hoeffding bound. We can first amplify the error probability so that the probability of error is less than 2^{-n}. Then by the union bound, for each input length, there must be some fixed string r such that $M(x, r) = f(x)$ for each of the 2^n choices of x. Then we can use a circuit to hardcode this r and compute f in polynomial size.

We do not know whether **BPP** = **P** and this is a major open question. However, there have been some interesting conditional results. For example, work of Impagliazzo, Nisan and Wigderson has led to the following theorem:

Theorem 8. *If there is some function* $f \in$ **EXP** *such that for every constant* $\epsilon > 0$, *f cannot be computed by a circuit family of size* $2^{\epsilon n}$, *then* **BPP** = **P**.

The theorem is interesting because the assumptions don't seem to say anything about useful. The assumption is that there is a function that can be computed by exponential time turing machines but cannot be computed by subexponential sized circuits. This fact is cleverly leveraged to derandomize any randomized computation. The proof of this theorem is outside the scope of this course.

Schwartz-Zippel Lemma

Recall that a polynomial $p(x, y, z)$ is an expression of the form

$$14x^2y^5z^8 - 3x^3 + 17y^6z^3.$$

The degree of the polynomial is the maximum of the sums of the powers of the variables in any monomial. So in the last example, the degree is 15.

The Schwartz-Zippel Lemma turns out to be quite useful for randomized algorithms:

Lemma 9. *Let $p(x_1, \ldots, x_n)$ be a polynomial of degree d, such that p is not the 0 polynomial. Let S be any set of numbers, and let $a_1, \ldots, a_n$ be n random numbers drawn from S. Then $\Pr[p(a_1, \ldots, a_n) = 0] \le d/|S|$.*

Proof   We prove the lemma by induction on n. When $n = 1$, the theorem follows from the fact that any non-zero degree d polynomial in one variable has at most d roots. Thus $p(a) = 0$ only when a is a root, which happens with probability at most d.

For the general case. Let us write the polynomial in the form

$$p(x_1, \ldots, x_n) = x_n^\ell \cdot q(x_1, \ldots, x_{n-1}) + r(x_1, \ldots, x_n),$$

where here r is a polynomial in which the degree of x_n is at most $\ell - 1$. So we simply gather all the terms which have maximum degree in x_n.

Now let E_1 be the event that $p(a_1, \ldots, a_n) = 0$, and let E_2 be the event that $q(a_1, \ldots, a_{n-1}) = 0$. Then we have that

$$\begin{aligned}
\Pr[E_1] &= \Pr[E_1 \wedge E_2] + \Pr[E_1 \wedge \neg E_2] \\
&= \Pr[E_2] \cdot \Pr[E_2|E_1] + \Pr[\neg E_2] \cdot \Pr[E_1|\neg E_2] \\
&\le \Pr[E_2] + \Pr[E_1|\neg E_2].
\end{aligned}$$

By induction, since q is a degree $d - \ell$ polynomial, $\Pr[E_2] \le (d - \ell)/|S|$. Since after $x_1, \ldots, x_{n-1}$ are fixed in $\neg E_2$, we have that $p(a_1, \ldots, a_{n-1}, x_n)$ is a non-zero polynomial of degree ℓ, we have that $\Pr[E_1|\neg E_2] \le \ell/|S|$. Thus $\Pr[E_1] \le d/|S|$. ∎

Polynomial Identity Testing

One can ask whether there are interesting problems that are known to be in **BPP** but not known to be in **P**. Although there are many examples of problems for which the fastest algorithms are randomized (for example, primality testing), there are not so many examples for which the only known algorithm is randomized. A key such example is the problem of polynomial identity testing.

We are given an arithmetic circuit (namely a circuit that uses multiplication and addition gates). The goal is to determine whether the polynomial computed by the circuit is identically 0. There is a subtle issue here that needs to be clarified. Note that two different polynomials may compute the same function on a particular set of inputs.

For example, if the inputs are all binary, then $x_i^2 = x_i$ for any variable x_i. Indeed, if we changed the problem above to ask whether or not the arithmetic circuit computes the 0 function on binary inputs, then we obtain an **NP**-complete problem.

There is a simple randomized algorithm for identity testing. We pick random integers from a large enough set and evaluate the circuit on those inputs. If the circuit computes a non-zero polynomial, it can be shown that the output will be non-zero with high probability. To actually make this work, we need to make sure that evaluating the circuit can be done efficiently. Indeed the evaluation can easily compute a number that is as big as 2^{2^s} with a circuit of size s, which is too big to manipulate. It turns out that one can just do all the evaluations modulo a large random prime number p and obtain the same guarantees.

We do not know how to get a deterministic algorithm for this problem.