# *Lecture 13: Randomized Complexity Classes*

*Anup Rao*

*May 11, 2021*

## *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,

Here we use the fact that $\frac{1}{1-\epsilon} < 1 + \epsilon$.

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

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**.

We also have:

**Theorem 7.** **ZPP** $=$ **RP** $\cap$ *co***RP**.

**Proof** Suppose $f \in$ **ZPP**, via a randomized algorithm $M$ whose expected running time is $t(n)$. Consider the algorithm that simulates $M$ for $10t(n)$ steps, and outputs 0 if the simulation does not halt. Then clearly, the algorithm only makes an error if the correct answer is 1. On the other hand, the probability that running time of $M$ exceeds $10t(n)$ is at most $1/10$ (or else the expected running time would exceed $t(n)$. Thus we obtain an **RP** algorithm. The same idea (reversing the roles of 0 and 1) gives a *co***RP** algorithm.

For the other direction, suppose $f$ has an **RP** algorithm $M_1$ and a *co***RP** algorithm $M_0$. Then on input $x$ consider the algorithm that alternatively runs $M_0(x), M_1(x), M_0(x), \ldots$ until either $M_1(x)$ outputs 1, or $M_0(x)$ outputs 0. If $M_1(x) = 1$, then it must be that $f(x) = 1$. Similarly if $M_0(x) = 0$, it must be that $f(x) = 0$. In any case, one of these two algorithms will verify the value of $x$ in an expected constant number of runs. ■

**Theorem 8.** *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 9.** *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 useful about randomized computation. Moreover, most people might believe that the assumption is true given what we know about counting arguments. 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.