# *Lecture 15: Error reduction, Schwartz-Zippel, Polynomial identity testing*

*Anup Rao*

*May 14, 2024*

## *Error reduction*

The choice of the constant $2/3$ in the definition of **BPP** is not crucial, as the following theorem shows:

**Theorem 1** (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 $\Pr_r[M'(x,r) = f(x)] \geq 1 - 2^{-n^d}$.*

To prove the theorem, 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 $\Pr[\sum_{i=1}^n X_i \geq pn(1+\epsilon)] \leq 2^{-\epsilon^2 np/4}$.*

**Proof** of Theorem 1: 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,

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

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 3.** **BPP** $\subseteq$ **EXP**.

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

**Theorem 4.** **RP** $\subseteq$ **NP**.

**Theorem 5.** *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 6.** *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. 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 7.** *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* $\Pr[p(a_1,\ldots,a_n) = 0] \leq d/|S|$.

We shall the proof of this Lemma in the next lecture. Now, let us quickly discuss an application of the lemma.

## *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.