6.080/6.089 GITCS	1 April 2008

Lecture 14

Lecturer: Scott Aaronson *Scribe: Geoffrey Thomas*

Recap

Randomness can make possible computation tasks that are provably impossible without it. Cryptography is a good example: if the adversary can predict the way you generate your keys, you cannot encrypt messages. Randomness is also good for breaking symmetry and making arbitrary choices.

We also have randomized algorithms. For example, to determine the equality of two polynomials given in nonstandard form, e.g.,

$$(1+x)^2 = 1 + 3x + 3x^2 + x^3$$

pick a few random values and see if they evaluate to the same thing. Since two different polynomials of degree d can only be equal at up to d points per the Fundamental Theorem of Algebra, after evaluating the polynomials at very few values, we can know with high probability whether they are equal.

Can we "derandomize" any randomized algorithm, i.e., can we convert it into a deterministic algorithm with roughly the same efficiency? This (formalized below as **P** versus **BPP**) is one of the central open questions of theoretical computer science.

Useful Probability Formulas

- Union bound: $\mathbf{Pr}[A \vee B] \leq \mathbf{Pr}[A] + \mathbf{Pr}[B]$
- Linearity of expectation: $\mathbf{E}[X+Y] = \mathbf{E}[X] + \mathbf{E}[Y]$ whether or not X and Y are independent
- Markov's inequality:
$$\mathbf{Pr}[X \geq k\mathbf{E}[X]] \leq \frac{1}{k}$$
  This is true for any distribution X that takes only non-negative values. To prove it: suppose it were false. Then the contribution to $\mathbf{E}[X]$ from X greater than $k\mathbf{E}[X]$ would be so big as to increase the expectation.

Complexity

Can we formalize the concept of a problem that can be solved efficiently with a randomized algorithm? There are several ways to do this.

The complexity class **BPP** (Bounded-Error Probabilistic Polynomial-Time) is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a polynomial time algorithm $M(x, r)$ such that for all inputs x,

- if $x \in L$, then $M(x, r)$ accepts with probability $\geq \frac{2}{3}$

- if $x \notin L$, then $M(x,r)$ accepts with probability $\leq \frac{1}{3}$.

Here r is a random string of polynomial length.

Why $\frac{1}{3}$ and $\frac{2}{3}$? Well, they're just two nice numbers that are separated from each other. If you want more accurate probabilities, you can use *amplification*: run the algorithm many times and take the majority answer. By combining many noisy answers, you can compute a single more accurate answer. So intuitively, it shouldn't matter what probabilities we use to define **BPP**, since we can amplify any success probability to any other with a constant number of repetitions.

But can we be more precise, and bound how many repetitions are needed to amplify a given success probability p to another probability q? This is basically a statistics problem, involving the tails of binomial distributions. Computer scientists like to solve such problems using a rough-and-ready yet versatile tool called the *Chernoff bound.* For n fair coin flips $X_1 \ldots X_n \in \{0,1\}$, let $X = X_1 + \cdots + X_n$. By linearity of expectation, $\mathbf{E}[X] = \frac{n}{2}$. The Chernoff bound says that for all constants $a > 0$,

$$\mathbf{Pr}\left[\left|X - \frac{n}{2}\right| > an\right] \geq c_a^n$$

for some constant $c_a < 1$. In other words, if we repeat our **BPP** algorithm n times, the probability that the majority of the answers will be wrong decreases exponentially in n.

To prove the Chernoff bound, the key idea is to bound the expectation not of X, but of c^X for some constant c:

$$\begin{aligned}
\mathbf{E}\left[c^X\right] &= \mathbf{E}\left[c^{X_1+\cdots+X_n}\right] \\
&= \mathbf{E}\left[c^{X_1}\cdots c^{X_n}\right] \\
&= \mathbf{E}\left[c^{X_1}\right]\cdots\mathbf{E}\left[c^{X_n}\right] \\
&= \left(\frac{1+c}{2}\right)^n
\end{aligned}$$

Here, of course, we've made crucial use of the fact that the X_i's are independent. Now by Markov's inequality,

$$\begin{aligned}
\mathbf{Pr}\left[c^X \geq k\left(\frac{1+c}{2}\right)^n\right] &\leq \frac{1}{k} \\
\mathbf{Pr}\left[X \geq \log_c k + n\log_c \frac{1+c}{2}\right] &\leq \frac{1}{k}
\end{aligned}$$

We can then choose a suitable constant $c > 1$ to optimize the bound; the details get a bit messy. But you can see the basic point: as we increase $d := \log_c k$—which intuitively measures the deviation of X from the mean—the probability of X deviating by that amount decreases *exponentially* with d.

As a side note, can we amplify *any* difference in probabilities—including, say, a difference between $1/2 - 2^{-n}$ and $1/2 + 2^{-n}$? Yes, but in this case you can work out that we'll need an exponential number of repetitions to achieve constant confidence. On the other hand, so long as the inverse of the difference between the two acceptance probabilities is at most a polynomial, we can amplify the difference in polynomial time.

Other Probabilistic Complexity Classes

BPP algorithms are "two-sided tests": they can give errors in both directions. Algorithms like our polynomial-equality test can give false positives but never false negatives, and are therefore called “one-sided tests.” To formalize one-sided tests, we define another complexity class **RP** (Randomized Polynomial-Time). **RP** is the class of all languages $L \subseteq \{0,1\}^*$ for which there exists a polynomial time algorithm $M(x,r)$ such that for all inputs x,

- If $x \in L$, then $M(x,r)$ accepts with probability $\geq \frac{1}{2}$.
- If $x \notin L$, then $M(x,r)$ always rejects regardless of r.

The polynomial-*non*equality problem is in **RP**, or equivalently, the polynomial-equality problem is in co**RP**.

P is in **RP**, co**RP** and **BPP**. **RP** and co**RP** are in **BPP**, because we can just amplify an **RP** algorithm once and reject with probability $0 \leq \frac{1}{3}$ and accept with probability $\frac{3}{4} \geq \frac{2}{3}$. $\mathbf{RP} \cap co\mathbf{RP}$ is also called **ZPP**, for Zero-Error Probabilistic Polynomial-Time. It might seem obvious that $\mathbf{ZPP} = \mathbf{P}$, but this is not yet known to be true. For even given both **RP** and co**RP** algorithms for a problem, you might get unlucky and always get rejections from the **RP** algorithm and acceptances from the co**RP** algorithm.

RP is in **NP**: the polynomial certificate that some $x \in L$ is simply any of the random values r that cause the **RP** algorithm to accept. (Similarly, co**RP** is in co**NP**.) **BPP** is in **PSPACE** because you can try every r and count how many accept and how many reject.

Whether **BPP** is in **NP** is an open question. (Sure, you can generate a polynomial number of “random” numbers r to feed to a deterministic verifier, but how do you convince the verifier that these numbers are in fact random rather than cherry-picked to give the answer you want?) Sipser, Gács, and Lautemann found that $\mathbf{BPP} \subseteq \mathbf{NP}^{\mathbf{NP}}$, placing **BPP** in the so-called *polynomial hierarchy* ($\mathbf{NP}$, $\mathbf{NP}^{\mathbf{NP}}$, $\mathbf{NP}^{\mathbf{NP}^{\mathbf{NP}}}$, $\ldots$).

An even more amazing possibility than $\mathbf{BPP} \subseteq \mathbf{NP}$ would be $\mathbf{BPP} = \mathbf{P}$: that is, that every randomized algorithm could be derandomized. Nevertheless, the consensus on this question has changed over time, and today most theoretical computer scientists believe that $\mathbf{BPP} = \mathbf{P}$, even though we seem far from being able to prove it.

Of the several recent pieces of evidence that point toward this conclusion, let us mention just one. Consider the following conjecture:

> There is a problem solvable by a uniform algorithm in 2^n time, which requires c^n circuit size (for some $c > 1$) even if we allow nonuniform algorithms.

This seems like a very reasonable conjecture, since it is not at all clear why nonuniformity (the ability to use a different algorithm for each input size) should help in simulating arbitrary exponential-time Turing machines.

In 1997, Impagliazzo and Wigderson proved that if the above conjecture holds, then $\mathbf{P} = \mathbf{BPP}$. Intuitively, this is because you could use the hard problem from the conjecture to create a *pseudorandom generator*, which would be powerful enough to derandomize any **BPP** algorithm. We’ll say more about pseudorandom generators in the next part of the course.