CSE531: Computational Complexity Theory February 1, 2012

Lecture 7 Randomized Computations

Lecturer: Anup Rao

1 Randomized Computation

We have considered what happens to the power of Turing Machines when we allow them to make non-deterministic choices. A more realistic way to (potentially) give Turing Machines additional power is to allow them to make random choices. A randomized turing machine is a machine that in each step is allowed to toss a coin in order to make a decision about which transition to make.

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.

We define the class **BPP**, which is the set of functions that are computable by polynomial time randomized turing machines. 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 $f(x) = 1$. 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.

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

Theorem 1 (Error Reduction in **BPP**). *Suppose there is a randomized polynomial time machine M, a 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 $\Pr_r[M'(x,r) = f(x)] \geq 1 - 2^{-n^d}$.*

In order 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**.

We also have:

Theorem 5. **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 halts. 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 6. **BPP** $\in$ **P**/poly.

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 (one that I am personally very interested in). However, there have been some interesting conditional results. For example, work of Impagliazzo, Nisan and Wigderson has led to the following theorem:

Theorem 7. *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.

2 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.

3 Randomness vs non-determinism

Theorem 8. $\mathbf{BPP} \subseteq \mathbf{NP}^{\mathsf{SAT}}$.

Proof Suppose $f \in \mathbf{BPP}$. Let us first reduce the error of the probabilistic algorithm for f to 2^{-n}. Suppose the algorithm uses m random bits. Thus, we just need to be able to distinguish the case when $M(x,r)$ accepts $1 - 2^{-n}$ fraction of all m bit strings from the case when it accepts only 2^{-n} fraction of all m bit strings. Distinguishing the fractions 1 from 0 would be easy (just try a single string). Distinguishing the fractions 1 from < 1 can be done with a query to SAT. So we shall reduce to this case.

Let $u_1, \ldots, u_k\{0,1\}^m$ be k random m bit strings, where k will be chosen to be much smaller than 2^n. Then we have the following claims, where here $r \oplus u_i$ denotes the bitwise parity of the m-bit string r with the m-bit string u_i.

Claim 9. *If $f(x) = 1$, for every choice of $u_1, \ldots, u_k$, there exists some $r \in \{0,1\}^m$ such that $\bigwedge_i M(x, r \oplus u_i) = 1$.*

The claim following from the union bound. If you pick a random r, the probability that $M(x, r \oplus u_i)$ is incorrect is at most 2^{-n}. Thus the probability that any of them is wrong is at most $k2^{-n} < 1$.

In the other case, we have:

Claim 10. *If $f(x) = 0$, there exist choices $u_1, \ldots, u_k$, such that for every $r \in \{0,1\}^m$, $\bigvee_i M(x, r \oplus u_i) = 0$.*

For any fixed r, the probability that all choices of u_i fail to give the correct answer is at most 2^{-nk}. Thus, as long as $nk > m$, by the union bound some choice of u_i will work for all choices of r.

Our final algorithm in $\mathbf{NP}^{\mathsf{SAT}}$ is as follows. We start by guessing $u_1, \ldots, u_k$ (say $k = m^2$)to satisfy Claim 10. Then we use the SAT oracle to check whether or not there is an r that makes $M(x, r \oplus u_i)$ accept for some i.

■