CS 6743 **Lecture 17** [1] **Fall 2007**

1 Randomized Computation

Why is randomness useful? Imagine you have a stack of bank notes, with very few counterfeit ones. You want to choose a genuine bank note to pay at a store. However, suppose that you don't know how to distinguish between a "good" bank note and a "bad" one. What can you do? Well, if you pick a bank note at random, you will be lucky with high probability (here the probability of picking a good bank note is equal to the fraction of good bank notes in your stack).

Let's consider a more realistic example. Alice and Bob communicate over some channel. The communication is very expensive. Both Alice and Bob have an n-bit number. Alice has $a = a_1 \dots a_n$ and Bob has $b = b_1 \dots b_n$. They want to know if $a = b$.

Clearly, this check can be achieved with at most $n + 1$ communicated: Alice just sends her number a to Bob. Bob then compares a and b, and sends back to Alice one bit (1 if $a = b$, and 0 otherwise). If we allow only deterministic protocols, then this bound is the best one can get.

However, if we allow randomized protocols, we can do much better. Here, a randomized protocol may have Alice and Bob make a mistake (e.g., they may think that $a = b$ when in fact $a \neq b$), but this error should happen with very small probability over the random choices of Alice and Bob. Next, we'll describe a $O(\log n)$-communication protocol to check if $a = b$.

Alice and Bob will first find the smallest prime number p such that $n^2 \leq p \leq n^3$. (The high density of primes guarantees that there will be a prime between n^2 and n^3; in the interval $1, 2, \dots, m$, there are about $\Theta(m/\ln m)$ primes.) Then Alice picks a random element $r \in Z_p$, computes $A(r) = a_1 r^{n-1} + a_2 r^{n-2} + \cdots + a_n \mod p$, and sends to Bob the pair $(r, A(r))$. Upon receiving this pair, Bob will compute $B(r) = b_1 r^{n-1} + b_2 r^{n-2} + \dots b_n \mod p$ and test if $A(r) = B(r)$. If they are equal, Bob will send 1 to Alice (saying that he thinks that $a = b$); otherwise, Bob will send 0 to Alice (saying that he thinks $a \neq b$).

Observe that the amount of communication for the described protocol is at most $2|p|+1 \in O(\log n)$, by our choice of $p \leq n^3$. Now let's analyze the correctness. First, if $a = b$, then $A(x) = B(x)$ as polynomials, and hence, $A(r) = B(r)$ for any r. So in this case, Alice and Bob correctly decide that $a = b$ with probability 1.

In the case where $a \neq b$, we have that $A(x)$ and $B(x)$ are different polynomials of degree at most $n - 1$. So, their difference $C(x) = A(x) - B(x)$ is a *non-zero* polynomial of degree at most $n - 1$. The probability that Alice and Bob erroneously decide that $a = b$ is exactly $\mathbf{Pr}_{r \in Z_p}[C(r) = 0]$. Now, since $C(x)$ is of degree at most $n - 1$, it may have at most $n - 1$ roots, i.e., values r at which C is zero. So, we have

$$\mathbf{Pr}_{r \in Z_p}[C(r) = 0] \leq (n-1)/|Z_p| \leq n/n^2 = 1/n.$$

[1] This lecture is a modification of notes by Valentine Kabanets

So, in case $a \neq b$, Alice and Bob will decide that $a \neq b$ with probability at least $1 - 1/n$.

Note that by picking p to be a larger number, e.g., at least n^{100}, we can make the error probability of this protocol smaller than n^{-99}. The communication complexity remains $O(\log n)$.

2 Randomized Complexity Classes

A language $L \in \mathsf{RP}$ if there is a deterministic polytime TM $M(x, r)$ such that

1. for all $x \in L$, $\mathbf{Pr}_r[M(x, r) \text{ accepts}] \geq 1/2$, and
2. for all $x \notin L$, $\mathbf{Pr}_r[M(x, r) \text{ accepts}] = 0$.

Now, to decide L, on input x, a randomized algorithm may first flip $|r|$ random coins to compute a random string r, and then simulate $M(x, r)$. This randomized algorithm will run in polytime, and will be correct for all $x \notin L$ with probability 1, and will be correct for all $x \in L$ with probability at least $1/2$.

Note that if $L \in \mathsf{RP}$, then $L \in \mathsf{NP}$. So, we get that $\mathsf{RP} \subseteq \mathsf{NP}$. The class RP contains those languages that can be decided probabilistically with *one-sided* error: an algorithm may err on positive instances, but never on negative instances. Next, we define the class of languages decidable with two-sided error.

A language $L \in \mathsf{BPP}$ if there is a polytime DTM $M(x, r)$ such that

1. for all $x \in L$, $\mathbf{Pr}_r[M(x, r) \text{ accepts}] \geq 3/4$, and
2. for all $x \notin L$, $\mathbf{Pr}_r[M(x, r) \text{ accepts}] \leq 1/4$.

Note that now we allow a "small" probability of error even on inputs not in the language.

3 Reducing the error probability

Consider any $L \in \mathsf{RP}$. Let $M(x, r)$ be the corresponding polytime DTM. We design a new DTM $M'(x; r_1, \dots, r_l)$ such that $M'(x; r_1, \dots, r_l)$ accepts iff there is some $1 \leq i \leq l$ such that $M(x, r_i)$ accepts; in other words, M' simulates M for l independent random strings. We claim that the error probability of the new M' is at most 2^{-l}.

Indeed, if $x \notin L$, then $M(x, r_i)$ rejects for every i, and so M' also rejects for all sequences $r_1, \dots, r_l$. On the other hand, if $x \in L$, then $\mathbf{Pr}_{r_1,\dots,r_l}[M'(x; r_1, \dots, r_l) \text{ rejects}]$ is equal to $\prod_{i=1}^{l} \mathbf{Pr}_{r_i}[M(x, r_1) \text{ rejects}] \leq (1/2)^l$. So, in the case where $x \in L$, the TM M' accept for a fraction $1 - 2^{-l}$ of all random sequences $r_1 \dots r_l$.

This ability to reduce error probability allows us to prove the following.

Theorem 1. $\mathsf{RP} \subseteq \mathsf{BPP}$

Proof. By the argument above, for any $L \in \mathsf{RP}$, there is a DTM $M(x,r)$ such that, for every $x \in L$, $\mathbf{Pr}_r[M(x,r) \text{ accepts}] \geq 1 - 1/4 = 3/4$ — simply take $l = 2$ in the above-mentioned error reduction procedure. On the other hand, for every $x \notin L$. $\mathbf{Pr}_r[M(x,r) \text{ accepts}] = 0 < 1/4$. So, $L \in \mathsf{BPP}$. □

What about NP and BPP? Is one a subclass of the other? It is unknown! The general belief is that BPP = P, and therefore, BPP is a subset of NP. But, no unconditional result of that kind is known.

4 Reducing the error in BPP

Recall that a language L is in BPP if there is a deterministic polytime TM $M(x,r)$, such that for $x \in L$, $M(x,r)$ accepts at least $3/4$ of r's, and for $x \notin L$, $M(x,r)$ accepts at most $1/4$ of all r's. In each case, the probability that M makes a mistake (e.g., accepts $x \notin L$) is at most $1/4$. The choice of this error probability to be $1/4$ is rather arbitrary. As we show next, it is always possible to make the error probability exponentially small, by increasing the running time only slightly.

The idea is to choose l independent copies of a random string r, i.e., chose independently uniformly at random $r_1, r_2, \ldots, r_l$. Then simulate $M(x, r_i)$, for each $1 \leq i \leq l$, noting whether M accepts or rejects for each r_i. Finally, our new algorithm will accept if and only if the *majority* (i.e., $> l/2$) of random strings r_i's were accepted.

Intuitively, if $x \in L$, then each r_i has the probability at least $3/4$ of being accepted by $M(x, r_i)$. So, on average, M will accept at least $3/4$ of the strings in the list $r_1, r_2, \ldots, r_l$. It seems very unlikely that the actual number of accepted r_i's will deviate significantly from the expected number. (This is basically the Law of Large Numbers from Probability Theory.)

To make this intuition precise, we'll need the following theorem.

Theorem 2 (Chernoff bounds)**.** *Let $X_1, \ldots, X_n$ be independent identically distributed random variables such that $\mathbf{Pr}[X_i = 1] = p$ and $\mathbf{Pr}[X_i = 0] = 1 - p$, for some $0 \leq p \leq 1$. Consider the new random variable $X = \sum_{i=1}^{n}$ with expectation $\mu = pn$. Then, for any $0 < \delta \leq 1$* [2]*,*

$$\mathbf{Pr}[|X - \mu| > \delta\mu] < 2e^{-\delta^2\mu/3}.$$

The proof of this theorem can be found in many textbooks (e.g., Papadimitriou's book), and will not be proved in class.

Back to the analysis of our new algorithm that simulates $M(x,r)$ on l independent copies of random string r_i. Let's define a random variable X_i, for $1 \leq i \leq l$, such that $X_i = 1$ if $M(x, r_i)$ produces a correct answer (i.e., accepts if $x \in L$, and rejects if $x \notin L$), and $X_i = 0$ otherwise. Observe that, by definition of BPP, we have $\mathbf{Pr}[X_i = 1] = p \geq 3/4$. Let's analyze the probability that our new algorithm makes a mistake, i.e., that the majority of

[2]For large δ, the quadratic dependence on δ in the exponent of the right-hand side is not true — the exponent becomes only linear in δ.

the variables $X_1, \dots, X_l$ are 0. Let $X = \sum_{i=1}^{l} X_i$, and let $\mu = pl \geq (3/4)l$ be the expectation of X. We have

$$\mathbf{Pr}[X < l/2] \leq \mathbf{Pr}[X < (2/3)\mu] = \mathbf{Pr}[|X - \mu| > \mu/3].$$

By the Chernoff theorem, the last probability is at most $2e^{-\mu/27} \leq 2e^{-l/36}$, which is exponentially small in l, the number of times we run the original algorithm M.

Thus, by taking $l = \mathsf{poly}(n)$, where $n = |x|$, we can reduce the probability error of a new algorithm below an inverse exponential in n, while still running in polytime.

5 BPP and Small Circuits

The ability to reduce the error probability in BPP has a curious consequence that every language in BPP is computable by a family of polysize Boolean circuits.

Theorem 3. BPP $\subset$ P/poly

Remark: Note that P $\subset$ P/poly trivially, since a polytime deterministic TM can be simulated by a polysize Boolean circuit. However, it is not at all trivial to argue that every BPP algorithm can also be simulated by a small Boolean circuit. The problem seems to be: what do we do with random strings used by the BPP algorithm? Now, we'll prove the theorem, and thus explain what to do with all those random strings.

Proof. Consider an arbitrary $L \in$ BPP. Since we know how to reduce the error probability below any inverse exponential in the input size, we may assume that there is a deterministic polytime TM $M(x, r)$ such that, for every $x \in L$, $\mathbf{Pr}_r[M(x, r) = 1] > 1 - 2^{-2n}$, and for every $x \notin L$, $\mathbf{Pr}_r[M(x, r) = 1] < 2^{-2n}$, where $|x| = n$, and $|r|$ is some polynomial in n.

Now, $\mathbf{Pr}_r[\exists x \text{ s.t. } M(x, r) \text{ is wrong}] \leq 2^n \mathbf{Pr}_r[M(x, r) \text{ is wrong for a fixed } x] \leq 2^n 2^{-2n} = 2^{-n} \ll 1$. The first inequality is the so-called "union bound" saying that for any events $E_1, \dots, E_n$, $\mathbf{Pr}[E_1 \vee E_2 \dots E_n] \leq \sum_{i=1}^{n} \mathbf{Pr}[E_i]$. The second inequality uses the fact that M has error probability at most 2^{-2n} for every input of size n.

Hence, there must *exists* at least one string $\hat{r}$ such that $M(x, \hat{r})$ is correct *for every input* x of length n. We can use such a string $\hat{r}$ as an advice string, and then just simulate $M(x, \hat{r})$ on any given input x of length n. Thus, $L \in$ P/poly. □

6 BPP and P

It is a big open question whether every language in BPP can be decided in *deterministic* polytime, i.e., whether BPP = P. There is some evidence that this equality indeed holds. The reason is the following result.

Theorem 4 (Impagliazzo, Wigderson). *If* DTime($2^{O(n)}$) *contains a language L of circuit complexity $2^{\Omega(n)}$, then* BPP = P.

The proof of this theorem is too involved to be presented in the "Intro to Complexity" course.

Some remarks about the Impagliazzo-Wigderson theorem. First, note that the complete derandomization of BPP is possible if we can efficiently *deterministically* construct truth tables of Boolean functions of near-maximum circuit complexity (since the maximum circuit complexity of an n-variable Boolean function is about $2^n/n$). The condition that $L \in \mathsf{DTime}(2^{O(n)})$ is equivalent to saying that the truth table of some "hard" Boolean function in n variables can be produced deterministically in time $\mathsf{poly}(2^n)$, polynomial in the size of the truth table. Note the interplay of uniform and nonuniform complexities here: We want to produce efficiently *uniformly* (in time $\mathsf{poly}(2^n)$) truth tables of Boolean functions of high ($2^{\Omega(n)}$) *nonuniform* (i.e., circuit) complexity.

The second comment is that there is a tradeoff between circuit complexity of a language in EXP and the efficiency of derandomization. For example, Babai, Fortnow, Nisan, and Wigderson showed that if some language in EXP needs more than polynomial circuit size, then every language in BPP can be decided deterministically in subexponential time, i.e., time $\mathsf{DTime}(2^{n^\epsilon})$, for any $\epsilon > 0$.

Finally, there is an intimate relation between derandomizing BPP and proving circuit lower bounds. It is shown by Impagliazzo and Kabanets that in order to prove $\mathsf{BPP} = \mathsf{P}$, one would have to prove superpolynomial circuit lower bounds of some kind. Thus, derandomization and circuit complexity are inextricably connected.

7 BPP and PH

It is not known how BPP and NP are related to each other. They both belong to PSPACE, however. (For BPP, we just enumerate all random strings of polysize, counting how many of them are accepted by our randomized polytime algorithm. This can be done in polyspace.)

Actually, even more is known about BPP.

Theorem 5 (Sipser)**.** $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Proof. Since $\mathsf{BPP} = \mathsf{coBPP}$ (check this!), it suffices to prove that $\mathsf{BPP} \subseteq \Sigma_2^p$.

We'll follow the proof due to Lautemann. The idea is simple. Assume that a given $L \in \mathsf{BPP}$ is decided by a probabilistic TM M with error probability less than 2^n, where n is the input size. As we saw in the last lecture, we can always reduce the error probability to be that low. Fix an input x of size n. Consider the set of A of random strings r on which our TM M accepts x, i.e., $A = \{r \mid M(x, r) \text{ accepts}\}$. Let R be the set of all random strings r, for an input of size n.

There are two cases. Case I: $x \in L$. Then, by our assumption, $\frac{|A|}{|R|} > 1 - 2^{-n}$. We will consider *translations* of the set A: given a binary string s, where $|s| = |r|$, we define the set $A \oplus s = \{a \oplus s \mid a \in A\}$, where $a \oplus s$ means the bitwise XOR of the strings a and s. We will argue that, since A is big, there will be a small number of strings $s_1, \dots, s_k$, for $k = |r|$, such that

$$R = \cup_{i=1}^k A \oplus s_i,$$

i.e., translating the set A for k times will cover the entire set R.

Claim 6. *If $|A|/|R| > 1 - 2^{-n}$, then there exist strings $s_1, \ldots, s_k$, for $k = |r|$, such that $R = \cup_{i=1}^{k} A \oplus s_i$.*

Proof of Claim. The proof is an easy application of the *Probabilistic Method.* We'll show that a random collection of k strings will have the required property with non-zero probability, and so a desired collection of s_i's certainly exists.

So, let's pick a random sequence $s_1, \ldots, s_k$ uniformly and independently. Let $S = \cup_{i=1}^{k} A \oplus s_i$. For a fixed string $r \in R$, we have

$$\mathbf{Pr}[r \notin S] = \prod_{i=1}^{k} \mathbf{Pr}[r \oplus s_i \notin A] < (2^{-n})^k = 2^{-nk}.$$

Hence, applying the "union bound",

$$\mathbf{Pr}[\exists r \in R \text{ such that } r \notin S] \leq |R| 2^{-nk} = 2^k 2^{-nk} \ll 1.$$

It follows that a randomly chosen sequence $s_1, \ldots, s_k$ is good with probability $1 - 2^{-nk+k} > 0$, and hence a good sequence exists. $\square$

Now, in case II: $x \notin L$. We have $|A|/|R| < 2^{-n}$. We claim that there is *no* sequence $s_1, \ldots, s_k$ such that $R = \cup_{i=1}^{k} A \oplus s_i$ in this case. The proof is very simple: $| \cup_{i=1}^{k} A \oplus s_i| \leq k|A| < \frac{k}{2^n}|R| \ll |R|$. So, we'll never be able to cover the set R by few "translated" copies of the small set A.

To summarize, what we proved above is the following: $x \in L$ iff there exist $s_1, \ldots, s_k$ such that for every $r \in R$ at least one of $M(x, r \oplus s_i)$ accepts. But this is exactly a Σ_2^p formula. Hence, $L \in \Sigma_2^p$. $\square$

8 Example of a BPP language

8.1 Polynomial Identity Testing

Suppose we are given two arithmetic formulas $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ with integer coefficients. We want to know whether these formulas are equivalent, i.e., whether $f \equiv g$ as polynomials. One way to check this is to write out all the coefficients of the polynomials f and g, and compare them one by one. However, there may be exponentially many such coefficients, and so this approach will result in a highly inefficient algorithm.

A better way to solve this problem is by using a randomized algorithm. We simply pick random values to the variables $x_1, \ldots, x_n$, and check if the two polynomials agree on these values. If the two polynomials are equivalent, then they will evaluate to the same value on any given point. If they are different polynomials, then it's very unlikely that they will agree on a random point. To formalize this, we need the following lemma. Recall that the degree of a monomial $x_1^{d_1} \ldots x_n^{d_n}$ is $d_1 + \cdots + d_n$; the total degree of a polynomial f is the maximum degree of a monomial of f.

Lemma 7 (Schwartz-Zippel, and many others)**.** *Let $f(x_1, \ldots, x_n)$ be a non-zero polynomial over a field F, such that the total degree of f is d. Let $S \subseteq F$ be a finite subset of field elements. Then*

$$\mathbf{Pr}_{r_1,\ldots,r_n \in S}[f(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Proof. By induction on the number of variables n. Base case of $n = 1$ is easy: a univariate degree d non-zero polynomial over a field can have at most d roots. Hence, a random point r is a root with probability at most $d/|S|$.

Assume the statement is true for k variables, and let's prove it for $k+1$ variables. Express a polynomial $f(x_1, \ldots, x_{k+1})$ as a polynomial $f_1(x_{k+1})$, whose coefficients are polynomials in $x_1, \ldots, x_k$ (by factoring out the expressions x_{k+1}^i for $1 \leq i \leq d$). Let d_1 be the degree in x_{k+1} of f_1. Let $p(x_1, \ldots, x_k)$ be the coefficient at $x_{k+1}^{d_1}$ in f_1; that is, $f_1(x_{k+1}) = x_{k+1}^{d_1} p(x_1, \ldots, x_k) + \ldots$. Note that the total degree of p is at most $d - d_1$, since d is the total degree of f. We have $\mathbf{Pr}_{r_1,\ldots,r_{k+1}}[f(r_1, \ldots, r_{k+1}) = 0] =$

$$\mathbf{Pr}[f(r_1, \ldots, r_{k+1}) = 0 \mid p(r_1, \ldots, r_k) \neq 0] * \mathbf{Pr}[p(r_1, \ldots, r_k) \neq 0] \tag{1}$$
$$+ \mathbf{Pr}[f(r_1, \ldots, r_{k+1}) = 0 \mid p(r_1, \ldots, r_k) = 0] * \mathbf{Pr}[p(r_1, \ldots, r_k) = 0]. \tag{2}$$

We can upperbound this sum as follows.

$$\begin{aligned}
&\mathbf{Pr}[f(r_1, \ldots, r_{k+1}) = 0 \mid p(r_1, \ldots, r_k) \neq 0] * \mathbf{Pr}[p(r_1, \ldots, r_k) \neq 0] \leq \\
&\mathbf{Pr}[f(r_1, \ldots, r_{k+1}) = 0 \mid p(r_1, \ldots, r_k) \neq 0] \leq \\
&\mathbf{Pr}_{r_{k+1}}[f_1(r_{k+1}) = 0],
\end{aligned}$$

where $f_1(r_{k+1})$ is obtained after substituting the values $r_1, \ldots, r_k$ for $x_1, \ldots, x_k$. Note that after such substitutions, we obtain a univariate polynomial of degree at most d_1. Hence,

$$\mathbf{Pr}_{r_{k+1}}[f_1(r_{k+1}) = 0] \leq d_1/|S|.$$

For the second summand, we have

$$\begin{aligned}
&\mathbf{Pr}[f(r_1, \ldots, r_{k+1}) = 0 \mid p(r_1, \ldots, r_k) = 0] * \mathbf{Pr}[p(r_1, \ldots, r_k) = 0] \leq \\
&\mathbf{Pr}[p(r_1, \ldots, r_k) = 0] \leq \\
&(d - d_1)/|S|,
\end{aligned}$$

where the last inequality is by the Inductive Hypothesis.

Putting everything together, we get

$$\mathbf{Pr}[f(r_1, \ldots, r_{k+1}) = 0] \leq d_1/|S| + (d - d_1)/|S| = d/|S|,$$

as promised. □