18.405J/6.841J Advanced Complexity Theory	March 1, 1998
Lecture 7	
Lecturer: Madhu Sudan	*Scribe: Adam R. Klivans*

1 Randomness in Computation

From a complexity theoretic perspective, a computational problem is thought to be tractable if it can be solved in deterministic polynomial time. One can wonder, however, what class of problems can be solved in polynomial time if we are allowed access to a source of randomness. Since many researchers feel that we have random (or pseudorandom) sources at our disposal, this class may more appropriately capture our notion of tractability. Before we state any definitions, we provide an illustrative example of the usefulness of randomness.

1.1 Branching Programs

Definition 1 *A* **Branching Program** *is a directed acyclic graph with the following properties:*

1 There is a start node of in-degree 0.

2 There is a reject and an accept node both with out-degree 0.

3 Every other (interior) node has outdegree 2 and contains a label x_i.

We can talk about the set of strings of length n accepted by a branching program in the following manner: Fix some string s. Each interior node of our branching program contains a label x_i where i ranges from 1 to n. This node "branches" on the ith symbol of our string s. We begin at the start node of our program and look at the label. Without loss of generality we can assume it is x_1. If $x_1 = 1$, we move to the left child of this node. If $x_1 = 0$ we move to the right child. We continue this process until we reach either the accept or reject node. If we reach the accept state we say that this string s was accepted by this program.

Notice that this program completely defines a boolean function f on n variables. To evaluate the function we simply move down a branch in the tree according to the input. Let

$$\mathbf{NEQ_{BP}} = \{(A, B) \mid f_A \not\equiv f_B\}$$

be the set of all pairs of branching programs A and B whose induced boolean functions are not equal. This set is known to be **NP**-complete.

Definition 2 *A* **Read Once Branching Program** *is a branching program where for any path from the start node to the accept/reject node, any variable x_i appears at most once.*

We will prove the following theorem

Theorem 3 *Let*

$$\mathbf{NEQROBP} = \{(A, B) \mid A \text{ and } B \text{ are ROBPs and } f_A \not\equiv f_B\}$$

Membership in this set can be decided in randomized polynomial time.

The proof will use algebraic techniques. For every branching program we associate a multivariate polynomial. This polynomial on boolean settings to the variables will compute the same boolean function that the branching program computes. We will show that if two branching programs are not equal, then the difference of the two induced multivariate polynomials is not identically zero. Finally, we show how to test if a multivariate polynomial is identically zero in randomized polynomial time.

First, we need to show how to construct a polynomial from a braching program. We give a recursive construction as follows (working bottom up):

- The accept node is assigned 1. The reject node is assigned 0.
- Assume we are at some interior node labeled x_i and its two children get assigned, inductively, polynomials p_1 and p_0. Then the polynomial $p(x_1, \ldots, x_n)$ for this node equals $x_i p_1 + (1 - x_i) p_0$.

Lemma 4 *Two branching programs A and B are not equal if and only if the polynomials constructed as above, p_A and p_B are not equal.*

Proof One direction is obvious. If polynomial p_A and p_B are equal, then in particular they are equal on all boolean assignments to the variables. Thus, our two branching programs will be equal (since p_A and p_B represent the branching programs' behavior). If our branching programs A and B are equal, we need to show that the polynomial p_A and p_B are equal everywhere, not just on boolean inputs. Notice, however, that because our branching program A is read once, p_A is simply a sum of multilinear monomials corresponding to accepting paths. That is to say, if the assignment $x_1 = 1, x_5 = 1, x_6 = 1$ (all other variables are zero) results in an accepting path, then the monomial $x_1 x_5 x_6$ will appear as one of the "minterms" in p_A. But, notice that $A \equiv B$. Thus, both p_A and p_B have identical "minterm" expansions (they are both sums of identical multilinear monomials). Hence, p_A and p_B are equal ev! erywhere. ■

Now we need the following lemma by Schwarz (or Schwarz-Zippel):

Lemma 5 *Assume $p(x_1, \ldots, x_n)$ is a non-zero polynomial, where each variable x_i has degree at most $d > 0$. Let I be any domain of size $\geq nd$. Then*

$$Pr_{\{x_1, \ldots, x_n\} \in I^n}(p(x_1, \ldots, x_n)) = 0 \leq \frac{dn}{|I|}$$

Proof: We use induction on n. Let $n = 1$. A non-zero polynomial of degree $\leq d$ can have at most d roots, so when we choose x_1 from a domain of size $|I|$, we have at least $\frac{d}{|I|}$ chance to pick a non-root, exactly as the inequality says.

Now, the inductive step. Let $d \geq d_1 =$ highest degree of x_1 in p. And let $p_2(x_2, \ldots, x_n)$ be the coefficient in front of $x_1^{d_1}$. We have by inductive hypothesis that

$$Pr_{\{x_2, \ldots, x_n\} \in I^{n-1}}(p(x_2, \ldots, x_n) = 0) \leq \frac{d(n-1)}{|I|}$$

We have:

$$\begin{array}{cl}
Pr_{\{x_1, \ldots, x_n\} \in I^n} & (p(x_1, \ldots, x_n) = 0) \\
= & Pr_{\{x_1, \ldots, x_n\} \in I^n}(p(x_1, \ldots, x_n) = 0 | p_2(x_2, \ldots, x_n) = 0) \times Pr_{\{x_2, \ldots, x_n\} \in I^{n-1}}(p(x_2, \ldots, x_n) = 0) \\
+ & Pr_{\{x_1, \ldots, x_n\} \in I^n}(p(x_1, \ldots, x_n) = 0 | p_2(x_2, \ldots, x_n) \neq 0) \times Pr_{\{x_2, \ldots, x_n\} \in I^{n-1}}(p(x_2, \ldots, x_n) \neq 0) \\
\leq & 1 \times \dfrac{d(n-1)}{|I|} + \dfrac{d}{|I|} \times 1 = \dfrac{dn}{|I|}.
\end{array}$$

The first bound $\frac{d(n-1)}{|I|}$ was made by the inductive assumption for $n-1$, the second bound $\frac{d}{|I|}$ - by observing that as long as we know that $p_2(x_2,\ldots,x_n) \neq 0$, no matter what $x_2,\ldots,x_n$ are chosen, there can be at most $d_1 \leq d$ out of $|I|$ possible values of x_1 making $p(x_1,\ldots,x_n) = 0$, and so the above bound follows.

As a corollary we have the following: let $|I| \geq 2nd$, $p(x_1,\ldots,x_n), q(x_1,\ldots,x_n)$ be 2 polynomials, and each variable in either of them has maximum degree $\leq d$. If we set $r = p - q$ then Schwarz's lemma gives us the following:

1. If $r \equiv 0$ (i.e. $p \equiv q$), then

$$Pr_{\{x_1,\ldots,x_n\}\in I^n}(r(x_1,\ldots,x_n) \neq 0) = 0$$

2. If $r \not\equiv 0$ (i.e. $p \not\equiv q$), then

$$Pr_{\{x_1,\ldots,x_n\}\in I^n}(r(x_1,\ldots,x_n) \neq 0) \geq \frac{1}{2}$$

Now our algorithm for testing equivalence of two branching programs is almost complete. Fix a field F of size greater than $2n^2$. We convert our programs into polynomials and pick random points $x_1,\ldots,x_n \in F^n$. Then we evaluate the difference of our polynomials on this random assignment to the variables. If the polynomials are not equal, then with probability $\frac{1}{2}$ we will discover this. Otherwise, the difference will always evaluate to 0. We can repeat this process polynomially many times to strengthen our confidence that the two polynomials are indeed identical.

We do have to be careful about evaluating these polynomials. The size of the polynomial grows exponentially in the size of the branching program. We can, however, **evaluate** the polynomial given some assignment by working bottom up on our branching program and following the polynomial construction given above, substituting in the appropriate values.

1.2 Randomized Complexity Classes

We have seen that picking random field elements has allowed us to make a seemingly difficult problem relatively easy to solve (with a negligible trade-off in confidence). In fact, it is not known whether a deterministic polynomial time algorithm exists for determining if two read once branching programs are equivalent. It is an important and well studied problem to determine if all randomized algorithms have deterministic counterparts. We formally define the most well known randomized complexity classes below.

Definition 6 *BPP is the class of all languages L for which there is a nondeterministic polynomial time TM, M, whose computation branches all have the same length, and which has the following properties.*
If $x \in L$ then at least 2/3 of the branches accept.
If $x \notin L$ then at most 1/3 of the branches accept.

Alternatively, using probability, we could say that

When $x \in L$ then $Pr[M(x)$ accepts $] \geq 2/3$
When $x \notin L$ then $Pr[M(x)$ accepts $] < 1/3$

Note that by both definitions, M is guaranteed to halt in polynomial time, with all branches halting together.

For the many probabilistic algorithms which have only a one-sided error, we define another class.

Definition 7 *RP is the class of all languages L for which there is a nondeterministic polynomial time TM, M, whose computation branches all have the same length, and which has the following properties.*

When $x \in L$ then $Pr[M(x)$ accepts $] \geq 2/3$
When $x \notin L$ then $Pr[M(x)$ accepts $] = 0$

Actually, most languages in BPP are known to be in RP or $Co-RP$. For example, COMPOSITES is in both RP and $Co-RP$. The intersection of RP and $Co-RP$ is an interesting class also, and has its own name.

Definition 8 *ZPP is the class $RP \cap Co-RP$.*

The numbers $1/3$ and $2/3$ in the above definitions may seem somewhat arbitrary, and in fact they are. We can reduce the error probability of an algorithm almost as much as we like by simple repetition. We will discuss the robust nature of these classes more in the next lecture.