6.841/18.405J: Advanced Complexity Theory		February 25, 2001
	Lecture 06	
Lecturer: Madhu Sudan		*Scribe: Mike Alekhnovich*

PS submission suggestions:

- avoid if possible using PDF files. Tex or PS are just fine...

- try to choose a name of the file that can identify yourself. Certainly names like "ps1.ps" is not a great idea at all...

Brief content:

Today we will learn what randomized computation is, will define eight new complexity classes, and see several interesting randomized algorithms.

1 Randomized Computation

Why do we care about randomized complexity? Well, mostly physisists. It is their beleif that nature is random and the system always evaluates to increase the entropy. How does this assumption alter our notion of "physically realizable computing"? As usual, in order to study this resource formally we define a new complexity class(es) and then try to relate them to earlier defined ones. It doesn't really matter if they are physically realizable, as long as it is mathematically definable. However, it appears that this particular one is realizable, which is very exciting.

1.1 How to define?

In parallel to non-determinism there are two ways to introduce the resource formally:

- As a computation on Turing machine with a special state R which picks randomly between two other states, and jumps to one of them.
- As a computation on two input turing machine $M(x,y)$, where x is our "real" input and y is a "random" string.

We will give the second variant of the definition. Let L be a language and and M be two input turing machine. Following the terminology of logical proof theory we can define *completeness* and *soundness* of M recognizing L:

- **Completeness**. This is the minimal probability of M accepting a "true" word that belongs to the language. Formally
$$c = \inf_{x \in L} \Pr_y[M(x,y) \text{ accepts}]$$
- **Soundness**. This is the maximal probability that M accepts a "wrong" word that is not in the language. Formally
$$s = \sup_{x \notin L} \Pr_y[M(x,y) \text{ accepts}]$$

The case $s = 0$ is reffered as perfect soundness (no false positives), $c = 1$ as perfect completeness (no false negatives). Intuitively one can suggest 4 essentially different patterns of c and s that make the language L randomly tractable:

- $c > s$,

- $c > 0$, $s = 0$

- $c = 1$, $s < 1$

- $c = 1$, $s = 0$

They exactly correspond to the four randomized complexity classes (more precisely to the eight classes because one can bound the time and the space of the computaion).

1.2 Time bounded computation

In this case we require our machine $M(\cdot, \cdot)$ run in time polynomial in x (clearly this implies that we need only polynomial number of random bits, so we can assume that $|y| < |x|^{O(1)}$). According to the four patterns of s and c the following classes can be defined:

- BPP ($c = \frac{2}{3}$, $s = \frac{1}{3}$): BPP (stands for bounded error probability polynomial) is a most general randomized class. We admit our machine do both types of error with bounded probability.
- RP ($c = \frac{2}{3}$, $s = 0$): we admit only one-sided error of rejecting the "true" world.
- *coRP* ($c = 1$, $s = \frac{2}{3}$): the class dual to RP.
- ZPP ($c = 1$, $s = \frac{2}{3}$): ZPP (stands for zero-equal probability) is a class of randomized computations with expected polynomial time.

1). These classes are robust with respect to c and s. As we (probably) will show in the sequel, the particular choice of these constants doesn't play any role, we can choose them arbitrarily (as long as we preserve conditions $c = 1$, $s = 0$).
2). It is believed that classes P and BPP are very close to each other, it is even conjectured (although we are still far from proving it) that $P = BPP$. Sadly, we don't have a complete problem for BPP (in particular the natural idea with universal random machine doesn't seem to work). However this class contains a lot of other interesting problems.
3). How does ZPP work? Notice that in this case we don't admit any error, all we require of the machine is to run on average in polynomial time. It is an exercise to show that $ZPP = RP \cap coRP$. A natural example of a problem in ZPP is primality testing. It is not hard to show that $PRIMES \in coRP$. By a more sophisticated algorithm due to Adleman and Huang, $PRIMES \in RP$ as well. If we run both routins in parallel then we get a ZPP primality testing algorithm.

1.3 Space bounded computation

So far we have seen polynomially bounded randomized computation. What is our machine $M(\cdot, \cdot)$ is restricted to run in $LOG-SPACE$ instead? Well, there are four classes: BPL, RL, *coRL* and ZPL. They are defined analogously to time bounded classes, there are two catches though.

Catch 1: we should require M run in polynomial time.

Catch 2: in two input model we should provide only one-way access to the second input (the machine cannot return to previously given random bits).

1.4 Relationships with other classes

The following inclusions that relates randomized complexity with other classes are known:

- $RP \subseteq NP$
- $coRP \subseteq coNP$
- $BPP \subseteq PH$

- $BPP \subseteq P/poly$

It is open if $BPP \subseteq NP$, however BPP is contained in the higher levels of polynomial hierarchy. Later we will prove some of these inclusions. Let us now turn to some interesting problems in the defined classes.

1.5 RP and Polynomial Identity Testing

Polynomial over integers is a finite sum

$$P = \sum_{c_{i_1 i_2 \ldots i_n}} c_{i_1 i_2 \ldots i_n} \cdot x_1^{i_1} \cdot x_2^{i_2} \cdot \ldots \cdot x_n^{i_n}.$$

We can define *degree* of a single variable $\deg_{x_k}(P)$ as a maximal i for which P contains a term $(x_k^i \cdot \ldots)$. The total degree $\deg(P)$ of a polynomial P is the maximal sum $i_1 + i_2 + \ldots + i_n$ over all non-zero coefficients $c_{i_1 i_2 \ldots i_n}$.

Example. $P(x_1, x_2) = x_1^2 x_2^3 + 3x_1^3 + 4x_2^4$.

For this polynomial, $\deg_{x_1}(P) = 3$, $\deg_{x_2}(P) = 4$ and $\deg(P) = 5$.

Now we define *Polynomial Identity (to zero) Testing* problem:
Instance: *polynomial $A : \mathbf{Z}^n \to \mathbf{Z}$ represented as an oracle, an integer $d = \deg(A)$.*
Question: $\exists a_1, \ldots, a_n \; A(a_1, \ldots, a_n) \neq 0$?

Before we will give a randomized algorithm for this problem let us discuss some applications and consequences. As posed, the problem naturally contains in NP^A. In fact, one can show that it doesn't belong to P^A, thus this oracle separates P^A and RP^A.

Example of an application. *Let $M_{ij} = \sum c_k x_k$ be a matrix that contains polynomials as elements. Let $det(M)$ be its determinant calculated by standard rules (thus $det(M)$ is a polynomial over x). The question is $det(M) = 0$?*

We can consider $det(M)$ as an oracle, indeed for any specific value of x we can calculate $det(M)(x)$ by the Gaussian method. Together with an algorithm for polynomial identity this gives us a randomized algorithm that solves the problem.

Theorem. *Polynomial identity testing $\in RP$.*
Proof. We run the following algorithm:

1). *Set $m = 3d$*

2). *Pick $a_1, \ldots, a_n$ independently from $\{1, \ldots, m\}$.*

3). *If $A(a_1, \ldots, a_n) \neq 0$ then accept*
else reject

The question is why does this algorithm work? The soundness is trivially 0 (if there is no point for which $P \neq 0$, we will never accept.) The completeness is less trivial and appeals to the following useful lemma:

Lemma (Schwartz). *Let $P(x_1, \ldots, x_n)$ be a non-zero polynomial over an arbitrary field and S be a finite subset of domain of P. Then*

$$\Pr_{a_1, \ldots, a_n \in S} [P(a_1, \ldots, a_n) = 0] \leq \frac{d}{|S|}.$$

Proof by induction. If $n = 1$ then the statement of the lemma is just the classical theorem that degree d non-zero polynomial cannot have more than d roots. Assume that $n > 1$ and $P(x_1, ..., x_n)$ is a polynomial of degree d. Consider a variable x_n, let $d_n = \deg_{x_n}(P)$. Let us represent P in the form:

$$P(x_1, ..., x_n) = Q(x_1, ..., x_{n-1})x_n^{d_n} + R(x_1, ..., x_n)$$

so that $\deg(Q) \leq d - d_n$ and $\deg_{x_n}(R) < d_n$.

Let $a_1, ..., a_n$ be a randomly chosen subset of S. Let $g(x_n) = P(a_1, a_2, ..., a_{n_1}, x_n)$ be the univariate polynomial that results after we replace each $x_1, ..., x_{n-1}$ with $a_1, ..., a_{n-1}$ in P. Clearly $\deg g \leq d_n$. Define two "bad" events:

$$E_1 : \; Q(a_1, ..., a_{n-1}) = 0$$
$$E_2 : \; \bar{E}_1 \wedge g(x_n) = 0$$

Obviously if neither of them happens then $P(a_1, ..., a_n) \neq 0$. Thus we need to estimate the probability of the event $E_1 \vee E_2$. By the induction hypothesis, $\Pr[E_1] \leq \frac{d-d_n}{|S|}$. If the event E_1 does not occure then g can have at most d_n roots, thus we have $\Pr[E_2] \leq \frac{d_n}{|S|}$. Finally

$$\Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_2] \leq \frac{d}{|S|},$$

which finishes the proof.

Now, applying the lemma we get

$$c = \Pr_{a_1,...,a_n \in [1..m]}[A(a_1, ..., a_n) \neq 0] \geq \frac{2}{3},$$

and the theorem follows.

1.6 RL and Undirected S-T Connectivity

At the end we will briefly describe an example of a language in RL. Consider the following

Undirected S-T connectivity (USTCON) problem:
Instance: *Undirected graph G.*
Question: *Is there a path connecting s to t?*

We present an algorithm based on the random walk:

For n^4 steps do

Pick a random neighbor v of u

$u := v$

If $u = t$ then stop and accept

The algorithm runs random walk over the matrix of G for a while. If it reaches t, it is clearly connected. If after a great deal of time it has not succeeded, it rejects.

Despite the innocent simplicity of the algorithm, the analysis is by far not that simple. In fact, for a long time people had been thinking that USTCON is as complex as STCON, till the following beautiful result was proved by Aleliunas, Karp, Lipton, Lovász and Rackoff: For every connected graph with high probability the random walk will visit each node in $O(n^3)$ steps.