CIS 511: Theory of Computation **Mar 30, 2017**

Lecture 19: Randomized Complexity, Continued

Professor Sampath Kannan *Zach Schutzman*

NB: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

Randomized Polynomial (RP)

Last time we defined BPP as the class of algorithms with probabilistic polynomial time Turing machines which correctly decided membership in a language with probability at least $\frac{1}{3}$ (equivalently, strictly bounded from $\frac{1}{2}$). We showed last time that polynomial identity testing is in BPP (equivalently, deciding whether a polynomial is identically zero).

We also have the class PP, which is not very interesting. PP allows error as high as $\frac{1}{2} - \frac{1}{2^n}$. Things like SAT and beyond are in PP.

Definition 19.1 *A language L is in **Randomized Polynomial** (RP) if there is a probabilistic polynomial time Turing machine M such that given $x \in L$, M accepts with probability greater than or equal to $\frac{1}{2}$, and if $x \notin L$, M always rejects.*

We say RP has one-sided error, because we only make mistakes on strings in the language. If the algorithm accepts, we know for certain that the string is in the language. If it rejects, we cannot say for sure.

Let's look at polynomial non-identity testing again. Our algorithm from before always accepts when a polynomial is identically zero, but sometimes makes mistakes if the polynomial is not, but we got unlucky.

We know that $P \subseteq RP$ and $P \subseteq BPP$.

We can also do amplification on RP algorithms. We just run the algorithm k times and accept if any execution excepts. Therefore, $RP \subseteq BPP$ because we can run our RP algorithm twice to bring the error probability below $\frac{1}{3}$.

We also have $RP \subseteq NP$. To see this, consider $L \in RP$. If $x \in L$, then our RP machine M accepts x with probability at least $\frac{1}{2}$. There exists a sequence of 'coin flips' that cause M to accept on x. This serves as our certificate with which we can verify membership by deterministically simulating M on x with the specific sequence of random choices. Similarly, Co-$RP \subseteq$ Co-NP. Equality and/or proper containment is unknown.

Let's think about the problem of matrix multiplication checking. The language $L_{mmult} = \{\langle A, B, C\rangle | A, B, C$ are $n \times n$ matrices and $AB \neq C\}$. Matrix multiplication is already known to be in P ($O(n^{2.34\cdots})$). Therefore, verification is also in P. But, we can use randomization to verify in $O(n^2)$ steps.

Let's suppose $AB = D$ and we want to know if $D = C$. We can randomly sample cells in D to check if they match cells of C. We can find and compare a pair of cells in n steps. But we want to think of this in a worst-case or adversarial setting, where maybe C and D only differ at one entry. Then we need to check n^2 entries for n steps each, which isn't any better than solving the whole thing.

What we will do is, instead of randomly sampling, we just take a random vector v and check if $(AB)v = Cv$. This at least has the property that if AB equals C then we accept. We can compute $A(Bv)$ in $n^2 + n^2$ steps, *then do Cv in n^2 steps, so the process takes $O(n^2)$ steps. What is the probability of error? Suppose $D \neq C$. What is the probability that $Dv = Cv$?. C and D differ in at least one entry, call it $D_{ij} \neq C_{ij}$. If v is a binary vector chosen uniformly at random, suppose every value except v_j chosen ahead of time, which we will think of as a variable. When we multiply the ith row of D by v, we get some constant $c_1 + D_{ij}v_j$. Similarly, for the ith row of C, we get $c_2 + C_{ij}v_j$. We now want to think about choosing v_j. If $c_1 = c_2$, then the choice of $v_j = 1$ gives different answers. This occurs with probability $\frac{1}{2}$. On the other hand, if $c_1 \neq c_2$, then choosing $v_j = 0$ shows the difference, which also occurs with probability $\frac{1}{2}$. Hence this algorithm is in RP. We can improve this error probability by letting v have arbitrary, rather than binary entries.*

Communication Complexity

Let's think about a communication problem, originally called The Man on the Moon Problem. Let x be a string on a spaceship, of which Ground Control has a copy y. Before executing x, the spaceship wants to verify that their copy has not been corrupted, so it wants to communicate with Ground Control to confirm $x = y$. We are concerned about the communication complexity of this problem; the number of bits we need to communicate in order to accomplish this task. We will assume that communication is two-way, reliable, and interactive.

We'll use cryptographic convention and call the spaceship player Alice (A) and the Earth player Bob (B).

We will think of corruption of x as adversarial. If the length of x and y is n, to do this deterministically, we need to exchange at least n bits to verify the whole string. To see this, suppose $x_1 \neq x_2$ are two n-bit strings.

Claim 19.2 *We claim that any communication transcript on input (x_1, x_1) must be different from the transcript on (x_2, x_2).*

Proof:

Suppose the input is actually (x_1, x_2) and we assume for the sake of contradiction that the transcripts for (x_1, x_1) and (x_2, x_2) are the same.

Without loss of generality, assume that Alice transmits the first bit, assuming the protocol for (x_1, x_1). When Bob receives this, it's consistent with the transcript for (x_2, x_2), so Bob sends the next bit assuming the input is (x_2, x_2). This repeats and neither side has any information at any point to contradict their assumptions that they are working on (x_1, x_1) and (x_2, x_2), respectively.

We therefore need 2^n different transcripts to correspond to the 2^n possible n-bit strings they have. By the pigeonhole principle, at least one of these must have length 2^n, because the number of binary strings of length strictly less than 2^n is $2^n - 1$.

■