PRINCETON UNIVERSITY COS 522: COMPUTATIONAL COMPLEXITY

Lecture 6: Randomized Computation

Lecturer: *Sanjeev Arora* Scribe:*Manoj*

A randomized Turing Machine has a transition diagram similar to a nondeterministic TM: multiple transitions are possible out of each state. Whenever the machine can validly more than one outgoing transition, we assume it chooses randomly among them. For now we assume that there is either one outgoing transition (a deterministic step) or two, in which case the machine chooses each with probability 1/2. (We see later that this simplifying assumption does not effect any of the theory we will develop.) Thus the machine is assumed to have a fair random coin.

The new quantity of interest is the probability that the machine accepts an input. The probability is taken over the coin flips of the machine.

1 RP and BPP

DEFINITION 1 **RP** *is the class of all languages L, such that there is a polynomial time randomized TM M such that*

$$x \in L \Leftrightarrow \mathbf{Pr}[M \text{ accepts } x] \geq \frac{1}{2} \tag{1}$$

$$x \notin L \Leftrightarrow \mathbf{Pr}[M \text{ accepts } x] = 0 \tag{2}$$

We can also define a class where we allow two-sided errors.

DEFINITION 2 **BPP** *is the class of all languages L, such that there is a polynomial time randomized TM M such that*

$$x \in L \Leftrightarrow \mathbf{Pr}[M \text{ accepts } x] \geq \frac{2}{3} \tag{3}$$

$$x \notin L \Leftrightarrow \mathbf{Pr}[M \text{ accepts } x] \leq \frac{1}{3} \tag{4}$$

As in the case of NP, we can given an alternative definiton for these classes. Instead of the TM flipping coins by itself, we think of a string of coin flips provided to the TM as an additional input.

DEFINITION 3 **BPP** *is the class of all languages L, such that there is a polynomial time deterministic TM M and $c > 0$ such that*

$$x \in L \Leftrightarrow \mathbf{Pr}_{r \in \{0,1\}^{|x|^c}}[M \text{ accepts } x] \geq \frac{2}{3} \tag{5}$$

$$x \notin L \Leftrightarrow \mathbf{Pr}_{r \in \{0,1\}^{|x|^c}}[M \text{ accepts } x] \leq \frac{1}{3} \tag{6}$$

We make a few observations about randomized classed. First, **RP** $\subseteq$ **NP**, since a random string that causes the machine to accept is a certificate that the input is in the language. We do not know if **BPP** is in **NP**. Note that **BPP** is closed under complementation. We do not know of any complete problems for these classes. One difficulty seems to be that the problem of determining whether or not a given randomized polynomial time TM is an RP machine is undecidable.

1.1 Probability Amplification

In the definition of **BPP** above the actual values in the place of $\frac{2}{3}$ and $\frac{1}{3}$ are not crucial, as we observe below. These two numbers can be replaced by $1 - 1/2^n$ and $1/2^n$ without changing the class.

By repeatedly running a machine M accepting a **BPP** language L with probabilities as above, and taking the majority vote of the different decisions it makes, we can get a much better probability of a correct answer. Using a polynomial (in input size) number of repeated trials, we get an exponentially small probability of deciding the input wrongly. This follows from bounding the tail of a binomial distribution using Chernoff bounds.

2 Theorems about BPP

Now we show that all **BPP** languages have polynomial sized circuits.

THEOREM 1
BPP $\subseteq$ **P**/poly

PROOF: For any language $L \in$ **BPP** consider a TM M (taking a random string as additional input) which decides any input x with an exponentially low probability of error; such an M exists by the probability amplification arguments mentioned earlier. In particular let the probability $M(x, r)$ being wrong (taken over r) be $\leq 1/2^{(n+1)}$, where $n = |x|$. Say that an r is *bad* for x if $M(x, r)$ is an incorrect answer. Let t be the total number of choices for r. For each x, at most $t/2^{(n+1)}$ values of r are bad. Adding over all x, we conclude that at most $2^n \times t/2^{(n+1)}$ values of r are bad for some x. In other words, at least $t/2$ choices of r not bad for every x. In particular there is some one value of r for which $M(x, r)$ decides x correctly. We can hard-wire this r into a polynomial size circuit. Given an input x, the circuit simulates M on (x, r). Hence $L \in$ **P**/poly. $\square$

The next theorem relates **BPP** to the polynomial hierarchy.

THEOREM 2
BPP $\subseteq \mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$

PROOF: It is enough to prove that **BPP** $\subseteq \mathbf{\Sigma}_2^p$ because **BPP** is closed under complementation.

Suppose $L \in$ **BPP**, and M is a randomized TM for L, that uses m random bits such that $x \in L \Rightarrow \mathbf{Pr}_r[M(x, r) \text{ accepts }] \geq 1 - 2^{-n}$ and $x \notin L \Rightarrow \mathbf{Pr}_r[M(x, r) \text{ accepts }] \leq 2^{-n}$.

Fix an input x, and let S_x denote the set of r for which M accepts (x, r). Then either $|S_x| \geq (1 - 2^{-n})2^m$ or $|S_x| \leq 2^{-n}2^m$, depending on whether or not $x \in L$. We will show how to guess, using two alternations, which of the two cases is true.

Consider r as an element of $\mathbf{GF(2)}^m$. For a set $S \subset \mathbf{GF(2)}^m$ we define the *shift of S by* r_0 as $\{r + r_0 | r \in S\}$ (where the addition is as in $\mathbf{GF(2)}^m$, i.e., bit-wise XOR).

Suppose $x \in L$, so $|S_x| \geq (1 - 2^{-n})2^m$. We shall show that there are a small number of vectors such that the set of shifts of S by these vectors covers the whole space $\mathbf{GF(2)}^m$.

LEMMA 3
$\exists r_1, r_2, \ldots, r_k$, *where* $k = \lceil \frac{m}{n} \rceil + 1$ *such that* $\bigcup_{i=1}^{k} (S + r_i) = \mathbf{GF(2)}^m$.

PROOF: We shall show that $\mathbf{Pr}_{(r_1,r_2,\ldots,r_k)}[\bigcup_{i=1}^{k} (S + r_i) \neq \mathbf{GF(2)}^m] < 1/2$.

Let $z \in \mathbf{GF(2)}^m$. If r_1 is a random vector, then so is $z + r_1$. So $\mathbf{Pr}_{r_1}[z \notin S + r_1] = \mathbf{Pr}_{r_1}[z + r_1 \notin S] \leq 2^{-n}$. So, for each z,

$$\mathbf{Pr}_{r_1,r_2,\ldots,r_k}[z \notin \bigcup_{i=1}^{k} S + r_i] \leq 2^{-nk}$$

So, $\mathbf{Pr}_{r_1,r_2,\ldots,r_k}[\text{some } z \notin \bigcup_{i=1}^{k} S + r_i] \leq 2^{m-nk} < 1/2 < 1$. □

Now suppose $x \notin L$. Now $|S_x| \leq 2^{-n}2^m$. This is a small set, and the union of any k shifts of S can be of size at most $k2^{m-n} < 2^m$, and hence cannot equal $\mathbf{GF(2)}^m$.

Thus we have established that

$$\begin{aligned} x \in L \Leftrightarrow \exists r_1, r_2, \ldots, r_k \in \mathbf{GF(2)}^m \text{ such that} \\ \forall\, z \in \mathbf{GF(2)}^m \; M \text{ accepts } x \text{ using at least one of } z + r_1, z + r_2, \ldots, z + r_k \end{aligned} \tag{7}$$

Thus L is in $\mathbf{\Sigma}_2^p$. □

3 Model Independence

Now we argue that our specific assumptions about randomized TMs do not affect the classes **RP** and **BPP**.

We made the assumption that the Turing machine uses a two-sided fair coins for randomization. Now we shall see that it can use these to simulate a machine that uses k-sided coins. To simulate a k-sided fair coin do as follows: flip our two-sided coin for $\lceil \log_2 k \rceil$ times. If the binary number generated is in the range $[0, k-1]$ output it as the result, else repeat the experiment. This terminates in expected $O(1)$ steps, and the probability that it does not halt in $O(k)$ steps is at most 2^{-k}. Since the machine needs to simulate only poly(n) coin tosses, the probability that this simulation does not work can be made 2^{-n}, and so it does not substantially affect the probability of acceptance (which is something like 1/2 or 2/3).

Another issue that arises in real life is that one may not have an unbiased coin, and the probability it comes up heads is an unknown quantity p. But we can simulate a an unbiased coin as follows: flip the (biased) coin twice (we are assuming independence of different flips). Interpret 01 as 0 and 10 as 1; on a 00 or 11 repeat the experiment. The probability that we fail to produce a 0 or 1 is at most $1 - p^2 - (1 - p)^2$. Since p is constant, this failure probability is a constant. The expected number of repetitions before we produce a bit is $O(1)$.

4 Recycling Entropy

Randomness may be considered a resource, because it is likely that a random bit generator is much slower than the rest of the computer. So one would like to minimize the number of random bits used. One place where we can recycle random bits is while repeating a **BPP** or **RP** algorithm.

Consider running an **RP**-machine on some input x. If it accepts x we know x is in the language L. But if it rejects x, we repeat the experiment, up to k times. But to repeat the experiment each time, it is not necessary to acquire all new random bits all over again. Intuitively, we can "recycle" most of the randomness in the previous random string, because when the experiment fails to detect x being in L, all we know about the random string used is that it is in the set of *bad* random strings. Thus it still has a lot of "randomness" left in it.

DEFINITION 4 *If $d > 2$ and $\beta > 0$, a family of (d, β)-*expanders *is a sequence of graphs $\{G_n\}$ where G_n is a d-regular graph on i nodes, and has the property that every subset S of at most $n/2$ nodes has edges to at least $\beta |S|$ nodes outside S.*

Deep mathematics (and more recently, simpler mathematics) has been used to construct expanders. These constructions yield algorithms that, given n and an index of a node in G_n, can produce the indices of the neighbors of this node in $\text{poly}(\log n)$ time.

Suppose the **RP** algorithm uses m random bits. The naïve approach to reduce its error probability to 2^{-k} uses $O(mk)$ random bits. A better approach involving recycling is as follows. Pick the first random string as a node in an *expander graph* on 2^m nodes, take a random walk of length k from there, and use the indices of the nodes encounted in this walk (a total of mk bits) in your **RP** algorithm. Since the graph has constant degree, each step of the random walk needs $O(1)$ random bits. So a random walk of length k takes $m + O(k)$ random bits; m to produce the first node and $O(1)$ for each subsequent step. A surprising result shows that this sequence will be random enough to guarantee the probability amplification, but we will not give a proof here.

Of course, the same approach works for any randomized algorithm, such as a Monte Carlo simulation.