Lecture 7

In this lecture we will define the probabilistic complexity classes **BPP**, **RP**, **ZPP** and we will see how they are related to each other, as well as to other deterministic or circuit complexity classes. Further, we will describe a probabilistic algorithm for the primality testing.

1 Probabilistic complexity classes

First we are going to describe the probabilistic model of computation. In this model an algorithm A gets as input a sequence of random bits r and the "real" input x of the problem. The output of the algorithm is the correct answer for the input x with some probability.

Definition 1 *An algorithm A is called a polynomial time probabilistic algorithm if the size of the random sequence $|r|$ is polynomial in the input $|x|$ and $A()$ runs in time polynomial in $|x|$.*

If we want to talk about the correctness of the algorithm, then informally we could say that for every input x we need $\mathbf{Pr}[A(x,r) = \text{correct answer for} x] \geq \frac{2}{3}$. That is, for every input the probability distribution over all the random sequences must be some constant bounded away from $\frac{1}{2}$. Let us now define the class **BPP**.

Definition 2 *A decision problem L is in **BPP** if there is a polynomial time algorithm A and a polynomial $p()$ such that :*

$$\forall x \in L \quad \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] \geq 2/3$$

$$\forall x \notin L \quad \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] \leq 1/3$$

We can see that in this setting we have an algorithm with two inputs and some constraints on the probabilities of the outcome. In the same way we can also define the class **P** as:

Definition 3 *A decision problem L is in **P** if there is a polynomial time algorithm A and a polynomial $p()$ such that :*

$$\forall x \in L \quad : \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] = 1$$

$$\forall x \notin L : \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] = 0$$

Similarly, we define the classes **RP** and **ZPP**.

Definition 4 *A decision problem L is in* **RP** *if there is a polynomial time algorithm A and a polynomial $p()$ such that :*

$$\forall x \in L \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r)=1] \geq 1/2$$
$$\forall x \notin L \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r)=1] \leq 0$$

Definition 5 *A decision problem L is in* **ZPP** *if there is a polynomial time algorithm A whose output can be $0, 1, ?$ and a polynomial $p()$ such that :*

$$\forall x \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = ?] \leq 1/2$$

$$\forall x, \forall r \quad \textit{such that} \quad A(x,r) \neq ? \quad \textit{then} \quad A(x,r) = 1 \quad \textit{if and only if} \quad x \in L$$

2 Relations between complexity classes (or where do probabilistic classes stand?)

After defining these probabilistic complexity classes, let's see how they are related to other complexity classes and with each other.

Theorem 1
RP $\subseteq$**NP**

Proof: Suppose we have an algorithm for **RP**. Then this algorithm is also in **NP**. If $x \in L$ then there is a random sequence r, for which the algorithm answers yes. On the other hand, if $x \notin L$ then there is no witness. □

Most natural probabilistic algorithms belong to the class **RP**. We can also show that the class **ZPP** is more restricted than **RP**.

Theorem 2
ZPP $\subseteq$**RP**

Proof: We are going to convert a **ZPP** algorithm into an **RP** algorithm. The construction consists of running the **ZPP** algorithm and anytime it outputs ?, the new algorithm will answer 0. In this way, if the right answer is 0, then the algorithm will answer 0 with probability 1. On the other hand, when the right answer is 1, then the algorithm will give the wrong answer with probability less than $1/2$, since the probability of the **ZPP** algorithm giving the output ? is less than $1/2$. □

Another interesting property of the class **ZPP** is that it's equivalent to the class, which contains all languages for which there is an average polynomial time algorithm that always gives the right answer. More formally,

Theorem 3
A language L is in the class **ZPP** *if and only if L has an average polynomial time algorithm that always gives the right answer.*

PROOF: First let us clarify what we mean by average time. For each input x we take the average time of $A(x,r)$ over all random sequences r. Then for size n we take the worst time over all possible inputs x of size $|x| = n$. In order to construct an algorithm that always gives the right answer we run the **ZPP** algorithm and if it outputs a ?, then we run it again. Suppose that the running time of the **ZPP** algorithm is T, then the average running time of the new algorithm is:

$$T_{avg} = \frac{1}{2} \cdot T + \frac{1}{4} \cdot 2T + \ldots + \frac{1}{2^k} \cdot kT \approx O(T)$$

Now, we want to prove that if the language L has an algorithm that runs in polynomial average time $t(|x|)$, then this is in **ZPP**. What we do is run the algorithm for time $2t(|x|)$ and output a ? if the algorithm hs not yet stopped. It is straightforward to see that this belongs to **ZPP**. First of all, the worst running time is polynomial, actually $2t(|x|)$. Moreover, the probability that our algorithm outputs a ? is less than $1/2$, since the original algorithm has an average running time $t(|x|)$ and so it must stop before time $2t(|x|)$ at least half of the times. $\Box$

EXERCISE 1 **ZPP** =**RP** $\cap$**coRP**

Let us now prove the fact that **RP** is contained in **BPP**.

THEOREM 4
RP $\subseteq$**BPP**

PROOF: We will convert an **RP** algorithm into a **BPP** algorithm. In the case that the input x does not belong to the language then the **RP** algorithm always gives the right answer, so this is definitely in **BPP** as well. In the case that the input x does belong to the language then we need to boost the probability of a correct answer from at least $1/2$ to at least $2/3$.

More formally, let A be an **RP** algorithm for a decision problem L. We fix some number k and define the following algorithm:

$A^{(k)}$

input: x,
  pick $r_1, r_2, \ldots, r_k$
  if $A(x,r_1) = A(x,r_2) = \ldots = A(x,r_k) = 0$ **then return** 0
  else return 1

Let us now consider the correctness of the algorithm.In case the correct answer is 0 the output is always right, though in the case where the right answer is 1 the output is right except when all $A(x,r_i) = 0$.

$$\text{if} \quad x \notin L \quad \mathbf{Pr}_{r_1,\ldots,r_k}[A^k(x,r_1,\ldots,r_k) = 1] = 0$$

$$\text{if} \quad x \in L \quad \mathbf{Pr}_{r_1,\ldots,r_k}[A^k(x,r_1,\ldots,r_k) = 1] \geq 1 - \left(\frac{1}{2}\right)^k$$

It is easy to see that by choosing an appropriate k the second probability can go arbitrarily close to 1 and therefore become larger than 2/3, which is what is required by the definition of **BPP**. In fact, by choosing k to be a polynomial in $|x|$, we can make the probability exponentially close to 1. This enables us to change the definition of **RP** and instead of the bound of 1/2 for the probability of a correct answer when the input is in the language L, we can have a bound of $1 - \left(\frac{1}{2}\right)^{q(|x|)}$, for a fixed polynomial q. $\Box$

Let, now, A be a **BPP** algorithm for a decision problem L. Then, we fix k and define the following algorithm:

$A^{(k)}$

input: x,
  pick $r_1, r_2, \ldots, r_k$
  $c = \sum_{i=1}^{k} A(x, r_i)$
  if $c \geq \frac{k}{2}$ **then return** 1
  else return 0

In a **BPP** algorithm we expect the right answer to come up with probability more than 1/2. So, by running the algorithm many times we make sure that this slightly bigger than 1/2 probability will actually show up in the results. More formally let us define the Chernoff bounds.

THEOREM 5
(Chernoff Bound)
Suppose $X_1, \ldots, X_k$ are independent random variables with values in $\{0, 1\}$ and for every i, $\mathbf{Pr}[X_i = 1] = p$. Then:

$$\mathbf{Pr}[\tfrac{1}{k}\sum_{i=1}^{k} X_i - p > \epsilon] < e^{-\epsilon^2 \frac{k}{2p(1-p)}}$$

$$\mathbf{Pr}[\tfrac{1}{k}\sum_{i=1}^{k} X_i - p < -\epsilon] < e^{-\epsilon^2 \frac{k}{2p(1-p)}}$$

The Chernoff bounds will enable us to bound the probability that our result is far from the expected. Indeed, these bounds say that this probability is exponentially small in respect to k.

Let us now consider how the Chernoff bounds apply to the algorithm we described previously. We fix the input x and call $p = \mathbf{Pr}_r[A(x, r) = 1]$ over all possible random sequences. We also define the independent random variables $X_1, \ldots, X_k$ such that $X_i = A(x, r_i)$.

First, suppose $x \in L$. Then the algorithm $A^{(k)}(x, r_1, \ldots, r_k)$ outputs the right answer 1, when $\frac{1}{k}\sum_i X_i \geq \frac{1}{2}$. So, the algorithm makes a mistake when $\frac{1}{k}\sum_i X_i < \frac{1}{2}$.

We now apply the Chernoff bounds to bound this probability.

$$\mathbf{Pr}[A^{(k)}\text{outputs the wrong answer on } x] = \mathbf{Pr}[\tfrac{1}{k}\sum_i X_i < \tfrac{1}{2}]$$
$$\leq \mathbf{Pr}[\tfrac{1}{k}\sum_i X_i - p \leq -\tfrac{1}{6}]$$

since $p \geq \frac{2}{3}$.

$$\leq e^{-\frac{k}{72p(1-p)}} = 2^{-\Omega(k)}$$

The probability is exponentially small in k. The same reasoning applies also for the case where $x \notin L$. Further, it is easy to see that by choosing k to be a polynomial in $|x|$ instead of a constant, we can change the definition of a **BPP** algorithm and instead of the bound of $\frac{1}{3}$ for the probability of a wrong answer, we can have a bound of $2^{-q(|x|)}$, for a fixed polynomial q.

Next, we are going to see how the probabilistic complexity classes relate to circuit complexity classes and specifically prove that the class **BPP** has polynomial size circuits.

THEOREM 6
BPP $\subseteq$**SIZE**$(n^{O(1)})$

PROOF: Let L be in the class **BPP**. Then by definition, there is a polynomial time algorithm A and a polynomial p, such that for every input x

$$\mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = \text{wrong answer for } x] \leq 2^{-(n+1)}$$

This follows from our previous conclusion that we can replace $\frac{1}{3}$ with $2^{-q(|x|)}$. We now fix n and try to construct a family of circuits C_n, that solves L on inputs of length n.

LEMMA 7
There is a random sequence $r \in \{0,1\}^{p(n)}$ such that for very $x \in \{0,1\}^n$ $A(x,r)$ is correct.

PROOF: Informally, we can see that for each input x the number of random sequences r that give the wrong answer is exponentially small. Therefore, even if we assume that these sequences are different for every input x, their sum is still less than the total number of random sequences. Formally, let's consider the probability over all sequences that the algorithm gives the right answer for all input. If this probability is greater than 0, then the claim is proved.

$$\mathbf{Pr}_r[\text{for every } x, A(x,r) \text{ is correct}] = 1 - \mathbf{Pr}_r[\exists x, A(x,r) \text{ is wrong}]$$

the second probability is the union of 2^n possible events for each x. This is bounded by the sum of the probabilities.

$$\begin{aligned}
&\leq 1 - \sum_{x \in \{0,1\}^n} \mathbf{Pr}_r[A(x,r) \text{is wrong}] \\
&\leq 1 - 2^n \cdot 2^{-(n+1)} \\
&\leq \frac{1}{2}
\end{aligned}$$

□

So, we proved that at least half of the random sequences are correct for all possible input x. Therefore, it is straightforward to see that we can simulate the algorithm $A(\cdot,\cdot)$, where the first input has length n and the second $p(n)$, by a circuit of size polynomial in n.

All we have to do is find a random sequence which is always correct and build it inside the circuit. Hence, our circuit will take as input only the input x and simulate A with input x and r for this fixed r. Of course, this is only an existential proof, since we don't know how to find this sequence efficiently. □

In conclusion, let us briefly describe some other relations between complexity classes. Whether **BPP** $\subseteq$ **NP** or not is still an open question. What we know is that it's unlikely that **NP** is contained in **BPP**, since then by the previous result **NP** would have polynomial size circuits and hence by the result of Karp and Lipton the polynomial hierarchy would collapse.

3 A BPP algorithm for primality testing

We are going to describe a **BPP** algortihm for the problem of deciding whether a number is prime or not. Note that there are also algorithms for primality testing which belong to **RP** or **ZPP**.

We'll begin by some known facts of number theory.

DEFINITION 6 *A number $a \in \{0, \dots, n-1\}$ is a quadratic residue $(\bmod n)$ if there is a number r, such that $a \equiv r^2 (\bmod n)$.*

THEOREM 8
If n is a prime and a is a quadratic residue $(\bmod n)$ then a has exactly two square roots $(\bmod n)$ in $\{0, \dots, n-1\}$.

PROOF: We know that a has at least two roots, namely $r, -r$. We also know that there can't be more than two roots since $r^2 - a$ is a polynomial of degree 2. Hence, there are exactly two square roots. □

If now n is not a prime we can prove the following

THEOREM 9
If $n \geq 3$, not a prime power, and a is a quadratic residue, then a has at least four roots in $\{0, \dots, n-1\}$.

PROOF: Suppose $n = pq$, where p, q have no common factors. Then if r is a root of a, then we can construct the following four systems of equations.

$$\begin{aligned} x = r(\mod p) \quad &, \quad x = r(\mod q) \\ x = r(\mod p) \quad &, \quad x = -r(\mod q) \\ x = -r(\mod p) \quad &, \quad x = r(\mod q) \\ x = -r(\mod p) \quad &, \quad x = -r(\mod q) \end{aligned}$$

Using the Chinese Remainder Theorem, it is easy to see that each one of these systems has at least one solution which is different in each case. Each one of these solutions is also a root of a, hence a has at least four square roots. $\square$

We also know that there is a probabilistic polynomial time algorithm R that on input a prime n and a quadratic residue a, it finds two roots of a. If the input is not correct then the algorithm outputs arbitrarily. It is easy to see that this algorithm can only work for a prime, since otherwise it would also be an algorithm for factorization.

We are now ready to describe our algorithm for primality testing:

input: n (n odd and square free),
pick $r \in \{1, \dots, n-1\}$
  compute $R(n, r^2 \mod n)$
  if output of R not $\pm r$ **then** **return** "not prime"
  else return "prime"

In this algorithm, we select a random number, square it and ask for the roots. When n is a prime, then the algorithm R will find the two roots, which are $\pm r$ with high probability, hence our algorithm will answer "prime". On the other hand, when n is not prime, then since there are at least four roots, the algorithm R can return $\pm r$ with at most half probability. More precisely, the algorithm R is probabilistic and not exact, but there will still be a gap in the probabilities of the right and wrong answer. By amplifying this gap, we have that the algorithm is in **BPP**.