

Scribe - Lecture 2

Surya Raghav Baskar

April 1 2026

1 Introduction

We study about various complexity classes for randomized algorithms. The Turing machine is used for discussing complexity classes. The below section was not discussed in class, but may provide context for the remaining sections.

Definition 1. *A deterministic Turing machine is a quadruple*

$$M = (S, \Sigma, \delta, s).$$

Here S is a finite set of states, of which $s \in S$ is the machine's initial state. The machine uses a finite set of symbols, denoted Σ ; this set includes special symbols $BLANK$ and $FIRST$. The function δ is the transition function of the Turing machine, mapping

$$S \times \Sigma \rightarrow (S \cup \{HALT, YES, NO\}) \times \Sigma \times \{L, R, STAY\}.$$

The machine has three halting states: $HALT$ (the halting state), YES (the accepting state), and NO (the rejecting state); these are states, but formally they are not elements of S .

The input is written on the tape, unless specified otherwise. The machine starts in state s with its cursor on the first symbol of the input x ($FIRST$). The rest of the input is string of finite length from $(\Sigma \setminus \{BLANK, FIRST\})^*$. Left-most $BLANK$ defines the end of the string. The function δ based on the current state and symbol (s, α) , determines the next state s' , the symbol to be written β and the direction of motion ($L, R, STAY$).

If machine halts in YES or NO , then the input x is said to have been accepted or rejected respectively. The $HALT$ is for non-boolean functions, where the output is written onto the tape. An algorithm corresponds to a Turing machine that always halts.

A probabilistic Turing machine has an augmented ability to flip an unbiased coin in one step, so on any input x the probabilistic Turing machine accepts or rejects with some probability.

2 Complexity Classes

We will be discussing decision problems that will be treated as language recognition problem. For this fix an alphabet Σ , then Σ^* is the set of all possible strings over this alphabet. A language is $L \subseteq \Sigma^*$. The language recognition problem is to decide whether given x in Σ belongs to L . If we have an algorithm for specific L , then it means to have a underlying Turing machine that halts on *YES* for $x \in L$ and halts on *NO* for $x \notin L$. Any decision problem can be cast as a language recognition problem using a suitable encoding. We say an algorithm is efficient or poly-time, when we have an underlying Turing machine that halts in $n^{O(1)}$ steps, where n , is the length of input string.

Definition 2. A randomized algorithm is a deterministic algorithm that takes an additional random string as input $(A(x, r))$.

For a randomized algorithm we can define a general definition for efficiency or poly-time, if the runtime of $A(x, r)$ is $poly(|x|)$ for all (x, r) . Then for correctness we say the algorithm should be correct with probability atleast 0.5.

2.1 PP - Probabilistic Polynomial

Definition 3. A language $L \in PP$ iff there exists a randomized algorithm A running in worst case polynomial time such that for any $x \in \Sigma^*$

- $x \in L \implies \Pr[A(x) \text{ accepts}] > \frac{1}{2}$
- $x \notin L \implies \Pr[A(x) \text{ accepts}] < \frac{1}{2}$

The class PP is very powerful by having a weak definition. We have no bound on far from 1/2 the probabilities are, which could be exponentially small too (like 2^{-n}). In such cases we can't run the algorithm a polynomial number of independent trials to amplify the probability to a desired level, which would instead require exponentially many runs. The following theorem shows the power of PP class in relation to NP.

Definition 4. A language $L \in NP$, iff there exists a poly-time algorithm called the verifier $B(x, y)$ such that for any input $x \in \Sigma^*$

- $x \in L \implies \exists y \in \Sigma^*$ such that $B(x, y)$ accepts, where $|y|$ is bounded by a polynomial in $|x|$.
- $x \notin L \implies \forall y \in \Sigma^*$, $B(x, y)$ rejects.

Theorem 1. $PP \supset NP$

Proof. Let $L \in NP$. Then we have a polytime verifier $B(x, y)$. Then for input x , the witness size is $|y| \leq poly(|x|)$. Then total possible witnesses is $N = 2^{poly(|x|)}$. Let k be the number of witnesses for x , for which the algorithm accepts. This shows that $k \geq 1$ for $x \in L$ and $k = 0$ for $x \notin L$.

Now we construct the randomized algorithm $A(x, r)$ in this way for input x . Choose random witness y . Accept if $B(x, y)$ accepts, otherwise accept with probability $1/2 - \epsilon$ (where $\epsilon = 2^{-|y|-2}$)

Now if we look at the acceptance probability of $A(x, r)$

$$\begin{aligned} \Pr[A(x, r) \text{ accepts}] &= \Pr(\text{accept} \mid A \text{ accepts}) \cdot \Pr(A \text{ accepts}) \\ &\quad + \Pr(\text{accept} \mid A \text{ rejects}) \cdot \Pr(A \text{ rejects}). \\ &= k/N + (1 - k/N)(1/2 - \epsilon) \end{aligned}$$

we know $\epsilon = 1/4N$

Case 1: $x \in L$ ($k \geq 1$):

$$P \geq \frac{1}{2} - \frac{1}{4N} + \frac{1}{N} \left(\frac{1}{2} + \frac{1}{4N} \right) = \frac{1}{2} + \frac{1}{4N} + \frac{1}{4N^2} > \frac{1}{2}.$$

Case 2: $x \notin L$ ($k = 0$):

$$P = \frac{1}{2} - \epsilon = \frac{1}{2} - \frac{1}{4N} < \frac{1}{2}.$$

This shows that $A(x, r)$ randomized algorithm decides L with the success probability required for PP , so $L \in PP$. Thus $PP \supset NP$ \square

We have another intuitive result showing the power of PP class, that $PP \approx \#P$, where $\#P$ is counting the number of witnesses to NP problems.

2.2 BPP - Bounded-error Probabilistic Polynomial

Definition 5. A language $L \in BPP$ iff there exists a randomized algorithm A running in worst-case polynomial time such that for any $x \in \Sigma^*$:

- $x \in L \implies \Pr[A(x) \text{ accepts}] > \frac{2}{3}$
- $x \notin L \implies \Pr[A(x) \text{ accepts}] < \frac{1}{3}$

2.3 Probability amplification

Suppose a randomized algorithm A satisfies

- $x \in L \implies \Pr[A(x) \text{ accepts}] > c + \epsilon$
- $x \notin L \implies \Pr[A(x) \text{ accepts}] < c - \epsilon$

Then we can run the algorithm for $k = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ independent trials and accept if number of runs (say X) that accepts is $X > kc$. This makes the error $\leq \delta$. This way we can increase gap of BPP but not to 1 or 0.

2.4 RP - Randomized Polynomial

Definition 6. A language $L \in RP$ iff there exists a randomized algorithm A running in worst-case polynomial time such that for any $x \in \Sigma^*$:

- $x \in L \implies \Pr[A(x) \text{ accepts}] > \frac{1}{2}$
- $x \notin L \implies \Pr[A(x) \text{ accepts}] = 0$

Definition 7. The Solovay-Strassen primality test. If n is prime then

$$\forall_a a^{(n-1)/2} \equiv \frac{a}{n} \pmod{n}$$

where $\left(\frac{a}{n}\right)$ is the Jacobi symbol. For a single random a , there is at most 50% chance of the algorithm accepting a composite n as prime.

Theorem 2. $PRIMES \in CoRP$

Proof. By definition $L \in RP$ iff $L^c \in CoRP$. If L is the language of primes then L^c is the language of composites.

If we look at a modified Solovay-Strassen for recognizing the language of composites, we have the following. Pick a random $a \in \{1, \dots, n-1\}$, and test the following.

$$a^{(n-1)/2} \not\equiv \frac{a}{n} \pmod{n}$$

Then if n is composite, by Solovay-Strassen result we have at least 1/2 of a that satisfies. Then if n is prime, we have 0 a that satisfies. This is exactly the probability requirement for RP. Thus $COMPOSITES \in RP$, which in turn means $PRIMES \in CoRP$ \square

2.5 ZPP - Zero-error Probabilistic Polynomial

We can see that RP is the set of languages with a poly-time randomized algorithms that have zero False Positives. If we want a poly-time randomized algorithm with zero False Positives and False Negatives, then we exactly get the class P , as we can set the random string to any value and get deterministic poly-time algorithm.

If we loosen the constraint on runtime and ask for a expected poly-time runtime, with 100% correctness, we get the ZPP class.

Definition 8. A language $L \in ZPP$ iff there exists a randomized algorithm A running in expected polynomial time such that for any $x \in \Sigma^*$:

- $x \in L \implies A(x) \text{ accepts}$
- $x \notin L \implies A(x) \text{ rejects}$

Theorem 3. $ZPP = RP \cap CoRP$

Proof. We prove both directions

$$(\Rightarrow) ZPP \subseteq RP \cap CoRP$$

Let $L \in ZPP$, then we have an algorithm A with $\mathbb{E}[runtime] = R$, where R is some poly in $|x|$.

Now construct the following algorithm. Run A for $10R$ time. Before this time accept or reject according to output of A . If runtime passes this, then halt the machine and output reject always. Now based on our construction this a poly-time algorithm. Then for the correctness we have the following.

- $x \in L$, then $\Pr[runtime(A) > 10R] < \mathbb{E}[runtime(A)]/R < 1/10$ (by Markov). Then this means that A halts $9/10$ times and when it does halt, it accepts with zero error. Thus $\Pr[A(x) \text{ accepts}] \geq 9/10$
- $x \notin L$, then we either halt and correctly or fail to halt within $10R$ steps. In either case we reject. Thus $\Pr[A(x) \text{ accepts}] = 0$

The above argument shows $L \in RP$. We can have similar argument for $L \in CoRP$, where if we exceed the runtime, we halt and accept.

$$(\Leftarrow) RP \cap CoRP \subseteq ZPP$$

Now since $L \in RP$ and $L \in CoRP$, we create the following algorithm. Run A_{RP} and accept if it accepts, then run A_{CoRP} and reject if it rejects.

From definition of RP and $CoRP$ we know that

- $x \in L \implies A_{RP}$ accepts atleast $1/2$ the time and A_{CoRP} never rejects.
- $x \notin L \implies A_{CoRP}$ rejects atleast $1/2$ the time and A_{RP} never accepts.

The above argument shows that Algorithm has zero error.

For the runtime we have the following argument. For each step we $1/2$ chance of halting on the correct answer. Let X be the number of iterations to halt, then this is a geometric distribution with $p \geq 1/2$. Then the expected runtime is $\mathbb{E}[X] = 1/p \leq 2$. This shows that the expected runtime is poly, so the language $L \in ZPP$ \square

References

- [1] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.