MTH401: Theory of Computation

Probabilistic Turing Machines

Anoushka Sahay(200148), Priyanshu Maurya(200735), Vineet Kumar(201121), Yashanki Tonde(201151)

IIT Kanpur

April 1, 2025

Contents

1. Introduction
   - Probabilistic Turing Machines
   - Probabilistic Poly-Time TMs
   - Randomness vs Nondeterminism
   - PTM Computation
2. Complexity Classes
   - Two sided error : BPP
   - One sided error : RP, coRP
   - Relationship b/w coRP and BPP
   - Zero sided error : ZPP
   - Relationship b/w ZPP, RP and coRP
3. Amplification Lemmas
   - Characterization of BPP
   - Characterization of RP
   - Relationship b/w RP and BPP
4. Some examples of PTMs
   - Polynomial identity testing
   - Testing for perfect matching in a bipartite graph
   - Probabilistic Primality Testing

Introduction

Lets start by defining PTMs.

Definition

A *Probabilistic Turing machine*, abbreviated *PTM*, is a standard multitape Turing machine M with the following augmentation: M takes as input a random string $r \in \{0, 1\}^*$ as well as the usual input string $x \in \{0, 1\}^*$. Additionally, the machine comes equipped with a function $R : \mathbb{N} \to \mathbb{N}$ such that for each input (x, r) we have $|r| = R(|x|)$.

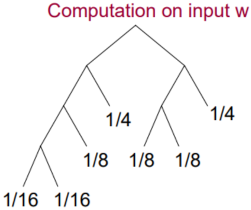
The definition of a probabilistic Turing machine (PTM) bears similarity to a non-deterministic Turing machine (NDTM) in that it allows multiple possible paths of computation. We will explore this connection further later.
Recall that for an NDTM M to recognize a language L, at least one computational path must correctly decide L. In the probabilistic case, we require a lower bound on the machine's ”success rate”.
There are various notions of computation by a PTM, depending on the allowed types of errors.

PTM

- New kind of NDTM, in which each nondeterministic step is a coin flip: has exactly 2 next moves, to each of which we assign probability $\frac{1}{2}$.
- Example: Computation on input *w*
    - To each maximal branch, we assign a probability:

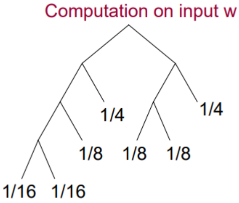
      $$\frac{1}{2} \times \frac{1}{2} \times \ldots \times \frac{1}{2}$$

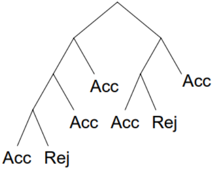
      (number of coin flips on the branch)
    - The probability of any branch is $\frac{1}{2^k}$ where k is the number of coin flips on the branch. For example, the probability of the two branches shown below is $\frac{1}{4}$ each.



PTM

- Probability of acceptance = $\sum_b$ an accepting branch $\Pr(b)$
- Probability of rejection = $\sum_b$ a rejecting branch $\Pr(b)$
- Example:
  - Add accept/reject information
  - Probability of acceptance = $\frac{1}{16} + \frac{1}{8} + \frac{1}{4} + \frac{1}{8} + \frac{1}{4} = \frac{13}{16}$
  - Probability of rejection = $\frac{1}{16} + \frac{1}{8} = \frac{3}{16}$
- We consider TMs that halt (either accept or reject) on every branch–deciders.
- So the two probabilities total 1.





Probabilistic Poly-Time TMs

Time Complexity:

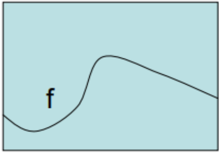
- Worst case over all branches, as usual.

Q: What good are probabilistic TMs?

- Random choices can help solve some problems efficiently.
- Good for getting estimates—arbitrarily accurate, based on the number of choices.

Example: Monte Carlo Estimation of Areas

- E.g., integral of a function f.
- Repeatedly choose a random point (x, y) in the rectangle.
- Compare y with $f(x)$.
- Fraction of trials in which $y \leq f(x)$ can be used to estimate the integral of f.



Probabilistic Poly-Time TMs

- **Q:** What does it mean to estimate a language?
- Each w is either in the language or not; what does it mean to ”approximate” a binary decision?
- **Possible answer:** For ”most” inputs w, we always get the right answer, on all branches of the probabilistic computation tree.
- **Or:** For ”most” w, we get the right answer with high probability.
- **Better answer:** For every input w, we get the right answer with high probability.
- **Definition:** A probabilistic TM decider M decides language L with error probability ε if:
  - $w \in L$ implies that $\Pr[M \text{ accepts } w] \geq 1 - \varepsilon$, and
  - $w \notin L$ implies that $\Pr[M \text{ rejects } w] \geq 1 - \varepsilon$.

Randomness vs Nondeterminism

We can impart probabilistic behavior to an NDTM by allowing it to randomly choose which of its two transition functions to employ at each step, with a lower bound on the success rate. This connection is summarized in the following proposition:

Proposition

Let M be an NDTM with transition functions δ_0, δ_1. Define a PTM M' such that, on input (x, r), M' computes M on x, where on the n-th step of computation M transitions via δ_0 if the n-th bit of r is 0, and via δ_1 if the n-th bit of r is 1. Then:

- M' computes a language L in bounded-error probabilistic time if and only if the proportion of computation paths of M which successfully decide each $x \in \{0,1\}^*$ is at least $\frac{2}{3}$, and
- M' computes a language L in randomized time if and only if the proportion of computation paths of M which successfully decide each $x \in L$ is at least $\frac{2}{3}$, and all computation paths successfully decide each $x \notin L$.

Thus we have made explicit the equivalence between the nondeterminism afforded by the two transition functions δ_0, δ_1, and the randomness provided by r.

PTM Computation

Definition

Let M be a PTM, $L \subseteq \{0,1\}^*$, and suppose that for any input (x, r), M halts in time $T(|x|)$ regardless of r.

1. We say M computes L in bounded-error probabilistic time $T(n)$ if for all $x \in \{0,1\}^*$ we have
$$\Pr[M(x,r) = L(x)] \geq \frac{2}{3}. \tag{1}$$
We denote the class of such languages by BPTIME$(T(n))$.
2. We say M computes L in randomized time $T(n)$ if for all $x \in \{0,1\}^*$ we have
$$x \in L, \implies \Pr[M(x,r) = 1] \geq \frac{2}{3} \tag{2}$$
$$x \notin L, \implies \Pr[M(x,r) = 1] = 1 \tag{3}$$
We denote the class of such languages by RTIME$(T(n))$.

Contd...

Remarks:

- The constant $\frac{2}{3}$ in both definitions is nearly arbitrary; what matters is that the success rate of the machine is bounded away from $\frac{1}{2}$. Note that we can always achieve a success rate of $\frac{1}{2}$ by simply guessing randomly.
- The key difference between the classes BPTIME and RTIME is their treatment of ”false positive” results. For a PTM M to compute L in randomized time, it must always correctly identify strings which are not in L. However, bounded-error probabilistic time permits what is called two-sided error, i.e., false positives and false negatives–so long as these are not too common.
- We can form the class coRTIME($T(n)$); it is not too hard to see that this corresponds to languages which are computable by PTMs as defined above, with the exception that we allow false positives and disallow false negatives. Under the definition below, this will lead to the class coRP which corresponds to languages recognizable in randomized polynomial time with no false negatives.

Two sided error : BPP

Definition

Language L is in BPP (Bounded-error Probabilistic Polynomial time) if there is a probabilistic polynomial-time TM that decides L with error probability 1/3.
$\mathrm{BPP} = \bigcup_c \mathrm{BPTIME}(n^c)$

Remark: The class BPP captures what we call probabilistic algorithms with two sided error. That is, it allows the machine M to output (with some small probability) both 0 when $x \in L$ and 1 when $x \notin L$.
Q: What's so special about $\frac{1}{3}$?

- Nothing. We would get an equivalent definition (same language class) if we chose ϵ to be any value with $0 < \epsilon < \frac{1}{2}$.
- We'll see this soon—Amplification Lemma.

One sided error : RP, coRP

Many probabilistic algorithms have the property of one-sided error. For example, if $x \notin L$, they will never output 1, although they may output 0 when $x \in L$. This is captured by the definition of RP.

Definition

Language L is in RP (Random Polynomial time) if there is a probabilistic polynomial-time TM that decides L, where:

- $w \in L$ implies that $\Pr[M \text{ accepts } w] \geq 1/2$,
- $w \notin L$ implies that $\Pr[M \text{ rejects } w] = 1$.

$\text{RP} = \bigcup_{c>0} \text{RTIME}(n^c)$.

- Always correct for words not in L.
- Might be incorrect for words in L—can reject these with probability up to $\frac{1}{2}$.
- Compare with nondeterministic TM acceptance:
  - $w \in L$ implies that there is some accepting path.
  - $w \notin L$ implies that there is no accepting path.

coRP

Definition

$coRP = \{L \mid L^c \in \text{ RP}\}$

- coRP contains the languages L that can be decided by a PPT-TM that is always correct for $w \in L$ and has error probability at most $\frac{1}{2}$ for $w \notin L$.
- That is, L is in coRP if there is a PPT-TM that decides L, where:
  $w \in L$ implies that Pr[M accepts w] = 1, and
  $w \notin L$ implies that Pr[M rejects w] $\geq \frac{1}{2}$.
- The class coRP captures one-sided error algorithms with the error in the "other direction" (i.e., may output 1 when $x \notin L$ but will never output 0 if $x \in L$).

Relationship between coRP and BPP

Theorem

$RP, coRP \subseteq BPP$

Proof: Observe RP, coRP $\subseteq$ BPP. This can be immediately seen from the definitions of the classes BPP, RP, and coRP.

Zero sided error : ZPP

- For a PTM M, and input x, we define the random variable TM_x to be the running time of M on input x.
- That is, $\Pr[TM_x = T] = p$ if with probability p over the random choices of M on input x, it will halt within T steps.
- We say that M has expected running time $T(n)$ if the expectation $\mathbb{E}[TM_x]$ is at most $T(|x|)$ for every $x \in \{0,1\}^*$.
- We now define PTMs that never err (also called ”zero error” machines).

Definition

The class ZTIME($T(n)$) contains all the languages L for which there is an expected-time $O(T(n))$ machine that never errs. That is,

- $x \in L \Rightarrow \Pr[M \text{ accepts } x] = 1$
- $x \notin L \Rightarrow \Pr[M \text{ halts without accepting } x] = 1$

We define ZPP $= \bigcup_{c>0}$ ZTIME(n^c).

Relationship between ZPP, RP and coRP

Theorem

$ZPP = coRP \cap RP$

Proof: (ZPP $\subseteq$ RP): Suppose $L \subseteq \{0,1\}^*$ is a language in ZPP. We need to show that $L \in$ RP $\cap$ coRP. Since $L \in$ ZPP, there is a probabilistic TM M with expected running time bounded by a polynomial function $q(|x|)$ that correctly decides the membership of every string $x \in \{0,1\}^*$.
We describe an RP algorithm (M_0) as follows:

1. On input x, run the machine M for $2q(|x|)$ many steps.
2. If M stops by this time, output whatever M outputs.
3. Otherwise, output 0.

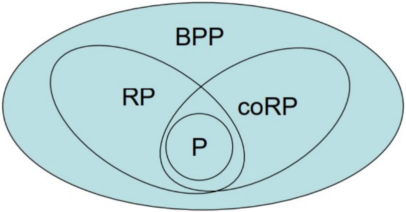
It is easy to see that this algorithm never makes a mistake when $x \notin L$. We wish to analyze the case when $x \in L$. Let $T(x)$ denote the running time of M on input x. Since $L \in$ ZPP, we have $\mathbb{E}(T(x)) \leq q(|x|)$, where the expectation is taken over the random choices of the machine M. By Markov's inequality, we have:

$$\mathbb{P}(T(x) \geq 2q(x)) \leq \frac{\mathbb{E}(T(x))}{2q(x)} \leq \frac{q(x)}{2q(x)} = \frac{1}{2}$$

Contd...

Since the probability that M has running time more than $2q(x)$ is less than $\frac{1}{2}$, it follows that $x \in L \Rightarrow \Pr(M_0(x) = 0) < \frac{1}{2}$. Thus, M_0 is an RP machine for the language L, and $L \in$ RP. Now, to show that $L \in$ coRP, we simply replace step 3 in the RP algorithm by ”otherwise output 1”. Then, it is easy to see that the algorithm never makes a mistake when $x \in L$. On the other hand, if $x \notin L$, the probability that M_0 makes a mistake is strictly less than $\frac{1}{2}$ (using the same argument), and hence $L \in$ coRP. Therefore, ZPP $\subseteq$ RP $\cap$ coRP. Similarly, we can show that RP $\cap$ coRP $\subseteq$ ZPP.

Relationship between Complexity Classes



Amplification Lemma

Lemma

Let ϵ be a fixed constant such that $0 \leq \epsilon < \frac{1}{2}$. Then for any polynomial $p(n)$, a PPT-TM M_1 that operates with error probability ϵ has an equivalent PPT-TM M_2 that operates with an error probability of ϵ' such that $0 \leq \epsilon' < \frac{1}{2}$ and $\epsilon' = 2^{-p(n)}$.

Proof Idea: M_2 simulates M_1 by running it a polynomial number of times and taking the majority vote of the outcomes. The probability of error decreases exponentially with the number of runs of M_1 made.
Consider the case where $\epsilon = \frac{1}{3}$. It corresponds to a box that contains red and blue balls. We know that $\frac{2}{3}$ of the balls are of one color and that the remaining $\frac{1}{3}$ are of the other color, but we don't know which color is predominant. We can test for that color by sampling 100 balls at random to determine which color comes up most frequently. Almost certainly, the predominant color in the box will be the most frequent one in the sample.
The balls correspond to branches of $M_1' s$ computation: red to accepting and blue to rejecting. M_2 samples the color by running M_1. A calculation shows that M_2 errs with exponentially small probability if it runs M_1 a polynomial number of times and outputs the result that comes up most often.

Proof of Amplification Lemma

Given *TM* M_1 which decides a language with an error probability $\epsilon < \frac{1}{2}$ and a polynomial $p(n)$, we construct a *TM* M_2 that decides the same language L with an error probability of $2^{-p(n)}$.

```
$M_2$ = On input $x$:
    1. Calculate $k$ (see analysis below).
    2. Run $2k$ independent simulations of $M_1$ on input $x$.
    3. If most runs of $M_1$ accept, then accept ; otherwise, reject.
```

We bound the probability that M_2 gives wrong answer on an input x. Stage 2 yields a sequence of $2k$ results from simulating M_1, each result either correct or wrong. If most of these results are correct, M_2 gives the correct answer. We bound the probability that at least half of these results are wrong.

Let S be any sequence of results that M_2 might obtain in stage 2. Let P_S be the probability that M_2 obtains S. Suppose that S has c correct results and w wrong results, so $c + w = 2k$. If $c \leq w$ and M_2 obtains S, then M_2 outputs incorrectly. We call such an S a *bad sequence*. Let ϵ_x be the probability that M_1 is wrong on x. If S is any bad sequence, then $P_S \leq (\epsilon_x)^w(1-\epsilon_x)^c$, which is at most $\epsilon^w(1-\epsilon)^c$ because $\epsilon_x \leq \epsilon < \frac{1}{2}$ so $\epsilon_x(1-\epsilon_x) \leq \epsilon(1-\epsilon)$, and because $c \leq w$. Furthermore, $\epsilon^w(1-\epsilon)^c$ is at most $\epsilon^k(1-\epsilon)^k$ because $k \leq w$ and $\epsilon < 1-\epsilon$.

Contd...

Summing P_S for all bad sequences S gives the probability that M_2 outputs incorrectly. We have at most 2^{2k} bad sequences because 2^{2k} is the number of all sequences. Hence,
$Pr[M_2 \text{ outputs incorrectly on input } x] = \sum_{bad\ S} P_S \leq 2^{2k}\epsilon^k(1-\epsilon)^k = (4\epsilon(1-\epsilon))^k$
We have assumed $\epsilon < \frac{1}{2}$, so $4\epsilon(1-\epsilon) < 1$. Therefore, the above probability decreases exponentially in k and so does $M_2's$ error probability. To calculate a specific value of k that allows us to bound $M_2's$ error probability by 2^{-t} for any $t \geq 1$,

$$(4\epsilon(1-\epsilon))^k \leq 2^{-t}$$

Taking log on both sides,

$$klog_2 4\epsilon(1-\epsilon) \leq -t$$
$$-log_2 4\epsilon(1-\epsilon) \geq \frac{t}{k}$$

Let $\alpha = -log_2(4\epsilon(1-\epsilon))$, we get,

$$\alpha \geq \frac{t}{k}$$

Choose $k \geq \frac{t}{\alpha}$.
Then we obtain an error probability of $2^{-p(n)}$ within polynomial time.

Characterization of BPP

Theorem

Given a language L, $L \in BPP$ if and only if, for some ϵ, $0 \leq \epsilon < \frac{1}{2}$, there is a PPT-TM that decides L with error probability ϵ.

Proof:
$\Rightarrow$ If $L \in BPP$, then there is some PPT-TM that decides L with error probability $\epsilon = \frac{1}{3}$, which suffices.
$\Leftarrow$ If for some ϵ, a PPT-TM decides L with error probability ϵ, then by the Amplification Lemma, there is a PPT-TM that decides L with error probability $\frac{1}{3}$; this means that $L \in BPP$.

Amplification Lemma

For *RP*, the situation is a little different. If $w \in L$, then $Pr[M\ accepts\ w]$ could be equal to $\frac{1}{2}$. So after many trials, the majority would be just as likely to be correct or incorrect. But this isn't useless, because when $w \notin L$, the machine always answers correctly.

Lemma

Suppose M is a PPT-TM that decides language L, $0 \leq \epsilon < 1$, and

$w \in L$ implies $Pr[M\ accepts\ w] \geq 1 - \epsilon$.

$w \notin L$ implies $Pr[M\ rejects\ w] = 1$.

Then for any ϵ', $0 \leq \epsilon' < 1$, there exists M', another PPT-TM, that decides L with:

$w \in L$ implies $Pr[M\ accepts\ w] \geq 1 - \epsilon'$.

$w \notin L$ implies $Pr[M\ rejects\ w] = 1$.

Proof Idea:

M' : On input w:

- Run k independent trials of M on w. If any accept, then accept; else reject.
- Here, choose k such that $\epsilon^k \leq \epsilon'$.
- If $w \notin L$ then all trials reject, so M' rejects, as needed.
- If $w \in L$ then each trial accepts with probability greater than or equal to $1 - \epsilon$, so $Pr[at\ least\ one\ of\ the\ k\ trials\ accepts] = 1 - Pr[all\ k\ reject] \geq 1 - \epsilon^k \geq 1 - \epsilon'$.

Characterization of RP

Theorem

Given a language L, $L \in RP$ if and only if for some ϵ, $0 \leq \epsilon < 1$, there is a PPT-TM that decides L with:

$w \in L$ implies $Pr[M\ accepts\ w] \geq 1 - \epsilon$.
$w \notin L$ implies $Pr[M\ rejects\ w] = 1$.

RP vs BPP

Theorem

RP$\subseteq$ BPP

Proof:

Given language A such that $A \in RP$, get (by definition of RP) a PPT-TM M with:

$w \in L$ implies $Pr[M\ accepts\ w] \geq \frac{1}{2}$.

$w \notin L$ implies $Pr[M\ rejects\ w] = 1$.

Here, $\epsilon = \frac{1}{2}$

By Lemma, get another PPT-TM for A, with:

$w \in L$ implies $Pr[M\ accepts\ w] \geq \frac{2}{3}$.

$w \notin L$ implies $Pr[M\ rejects\ w] = 1$.

Here, $\epsilon = \frac{1}{3}$

$1 - \epsilon = \frac{2}{3}$

Implies $A \in BPP$, by definition of BPP.

Examples of PTMs

Polynomial Identity Testing

Let $\mathbb{F}$ be a finite field, let $n \in \mathbb{N}$, and suppose $p, q \in \mathbb{F}[x_1, \dots, x_n]$ are polynomials of degree $\leq d$. For this example, we are concerned with determining whether $p = q$. We note two facts:

- Since $p = q$ if and only if $p - q = 0$, it suffices to consider the decision problem of whether a single polynomial is equal to zero. For this reason, we refer to this problem as Polynomial Identity Testing (PIT).
- Representation matters greatly here. If a polynomial p is given by its coefficients, PIT is trivial since $p = 0$ if and only if each coefficient is zero. In complexity theory, polynomials are often represented by algebraic circuits, which are similar to Boolean circuits except with $\mathbb{F}$-valued inputs and gates replaced by the operations $+$, $-$ and $\times$. We will abstract this away by instead allowing “black-box access” to p; that is, we will only assume that we can efficiently compute p on any input.

The question of resolving PIT is central to algebraic complexity theory. Many problems, including finding perfect matches in bipartite graphs, are reducible to PIT. Moreover, it is conjectured, but not known, that there exists a deterministic polytime algorithm to decide PIT. However, we have an easy random algorithm

Polynomial Identity Testing

At the heart of this algorithm is the following fact, typically known as the Schwartz-Zippel Lemma,

Schwartz-Zippel Lemma

Let $p \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial of total degree $d \geq 0$. Let S be a finite subset of $\mathbb{F}$ and let $r_1, \ldots, r_n$ be randomly and independently chosen from S. Then

$$\Pr[p(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}. \tag{4}$$

Proof: The proof is by mathematical induction on n. For $n = 1$, as we know, p can have at most d roots. This gives us the base case. Now, assume that the theorem holds for all polynomials in $n - 1$ variables. We can then consider p to be a polynomial in x_1 by writing it as

$$p(x_1, \ldots, x_n) = \sum_{i=0}^{d} x_1^i p_i(x_2, \ldots, x_n). \tag{5}$$

Since p is not identically 0, there is some i such that p_i is not identically 0. Take the largest such i. Then $\deg p_i \leq d - i$, since the degree of $x_1^i p_i$ is at most d.

Polynomial Identity Testing

Now we randomly pick $r_2, \ldots, r_n$ from S. By the induction hypothesis,

$$\Pr[p_i(r_2, \ldots, r_n) = 0] \leq \frac{d-i}{|S|}. \tag{6}$$

If $p_i(r_2, \ldots, r_n) \neq 0$, then $p(x_1, r_2, \ldots, r_n)$ is of degree i (and thus not identically zero), so

$$\Pr[p(x_1, r_2, \ldots, r_n) = 0 \mid p_i(r_2, \ldots, r_n) \neq 0] \leq \frac{i}{|S|}. \tag{7}$$

If we denote the event $p(x_1, r_2, \ldots, r_n) = 0$ by A, the event $p_i(r_2, \ldots, r_n) = 0$ by B, and the complement of B by B^c, we have

$$\begin{aligned} \Pr[A] &= \Pr[A \cap B] + \Pr[A \cap B^c] = \Pr[B] \Pr[A \mid B] + \Pr[B^c] \Pr[A \mid B^c] \\ &\leq \Pr[B] + \Pr[A \mid B^c] \leq \frac{d-i}{|S|} + \frac{i}{|S|} = \frac{d}{|S|}. \end{aligned} \tag{8}$$

Polynomial Identity Testing

With the Schwartz-Zippel lemma in hand, we can easily solve PIT. Suppose we are given a polynomial $p(x_1, \ldots, x_n)$ of total degree d. Assume that $d < |S|$.

Algorithm

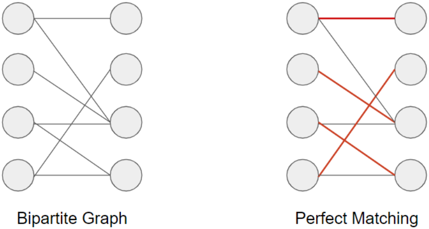
Input: Polynomial p of degree d in n variables
Output: Determine whether p is identically zero

```
1  Choose random inputs $(x_1, x_2, \ldots, x_n)$ from the domain $S$.
2  Compute $p(x_1, x_2, \ldots, x_n)$.
       If $p(x_1, x_2, \ldots, x_n) \neq 0$:
           return "Not identically zero"
       Else:
           return "Likely identically zero"
```

In the case $p(x_1, x_2, \ldots, x_n) \neq 0$ we will never make an error: we have conclusive proof that p is not identically zero. In the case when $p(x_1, x_2, \ldots, x_n) = 0$ the *Schwartz-Zippel lemma* shows that probability of error is at most $\frac{d}{|S|} < 1$.
We can decrease the probability of error to any desired level by repeating the algorithm multiple times. This is the amplification by independent trials trick.

Examples of PTMs

Testing for Perfect matching in a Bipartite Graph
If $G = (V_1, V_2, E)$ is the bipartite graph where $|V_1| = |V_2|$ and $E \subseteq V_1 \times V_2$, then a **perfect matching** is some $E_0 \subseteq E$ such that every node appears exactly once among the edges of E_0. Alternatively, we may think of it as a permutation σ on the set $\{1, 2, \ldots, n\}$ (where $n = |V_1|$) such that for each $i \in \{1, 2, \ldots, n\}$, the pair $(i, \sigma(i))$ is an edge. Here we describe a very simple randomized algorithm (due to Lovász) using the *Schwartz-Zippel lemma*.



Bipartite Graph

Perfect Matching

Figure: Illustration of how a perfect matching can be constructed on a bipartite graph.

Perfect Matcing

Consider the $n \times n$ bi-adjacency matrix X of G (where $n = |V_1| = |V_2|$). Rows of G are indexed by vertices in V_1, and columns by vertices in V_2 such that X_{ij} = 1 if $(i, j) \in E$ and 0 otherwise. The determinant of matrix $\det(X)$ is defined as

$$\det(X) = \sum_{\sigma \in S_n} (-1)^{\text{sign}(\sigma)} \prod_{i=1}^{n} X_{i,\sigma(i)} \tag{9}$$

where S_n is the set of all permutations of $\{1, 2, \ldots, n\}$. Notice that if $\det(X) \neq 0$, then we know that there must be some non-zero term, and thus G has a perfect matching. However, if $\det(X) = 0$, then it might be because G has no perfect matching, or non-zero terms cancel. We need to figure out which is the case.
To get around this, we associate a variable $x_{i,j}$ to each edge (i, j). In other words, consider X', a n-by-n matrix, where $X'_{i,j} = x_{i,j}$ if $(i, j) \in E$, and 0 otherwise. Then $\det(X')$ becomes a polynomial in variables $\{x_{i,j}\}_{(i,j) \in E}$. Notice that if G has a perfect matching M, then σ_M induces a monomial in $\det(X')$, The degree of $\det(X')$ is at most n, and $\det(X')$ is not identically zero. Otherwise, if G does not have a perfect matching, then $\det(X')$ is identically zero. This leads us to Lovász's randomized algorithm:

Perfect Matching

Lovász's Random Algorithm

1. Pick random values for x_{ij}'s from $[1, \ldots, 2n]$.
2. Substitute them in X and compute the determinant.
3. If the determinant is nonzero, output "accept" else output "reject".

Examples of PTM

Primality testing with the Miller-Rabin test
In primality testing we are given an integer $n \in \mathbb{N}$ and wish to determine whether or not it is prime.

Algorithm

1. If $n = 1$ or $n > 2$ is even: **return** "COMPOSITE".
2. Write $n - 1$ as $2^r \cdot d$, where d is odd
3. Pick a random integer a such that $1 < a < n - 1$;
4. Compute $x = a^d \bmod n$;
5. If $x \equiv \pm 1 \pmod{n}$ then go to step 3.
6. For $i = 1$ to $r - 1$, do:
   - Compute $x = x^2 \bmod n$;
   - If $x = -1$, go to step 3;
7. If none of the $r - 1$ repetitions results in a loop, then halt and output "COMPOSITE".

If the algorithm runs without halting, then the test concludes n is probably prime.

Primality Testing

The algorithm runs in polynomial time for any fixed k, and the success rate is exponential in k; therefore, the existence of this algorithm places primality testing in BPP.

Theorem

By repeating it k times, we get:

1. $n \in Primes \implies Pr[\text{accepts } n] = 1$
2. $n \notin Primes \implies Pr[\text{accepts } n] \leq (\frac{1}{2})^k$

Idea of Proof:
Some definations:
$PRIMES = \{n \mid n \in \mathbb{N} > 1$ and n cannot be factored as $qr,$ where $1 < q, r < n\}$
$COMPOSITES = \{n \mid n \in \mathbb{N} > 1$and n can be factored as $qr,$ where $1 < q, r < n\}$
The proof rests on some number-theoretic facts and Theorems about primes:

- **Fermat's Little Theorem:** If n is a prime and a is any integer not divisible by n, then $a^{n-1} \equiv 1 \pmod{n}$.

Primality Testing

- **Carmichael numbers:** A Carmichael number is an odd composite number n which satisfies Fermat's Little Theorem:

  $$a^{n-1} \equiv 1 \pmod{n}$$

  for every choice of a satisfying $\gcd(a, n) = 1$ (i.e., a and n are relatively prime) with $1 < a < n$.
- **Euler's Test:** If $a^{(n-1)/2} \not\equiv \pm 1 \pmod{n}$ for some a with $a \not\equiv 0 \pmod{n}$, then n is composite.
- **Fact 1:** Any non-Carmichael composite number fails at least half of all Fermat tests (for at least half of all values of a). So for any non-Carmichael composite, the Fermat's Little Theorem correctly identifies it as composite, with probability $\geq \frac{1}{2}$.
- **Fact 2:** For every Carmichael composite n, there is some $b \neq 1, -1$ such that $b^2 \equiv 1 \pmod{n}$ (that is, 1 has a nontrivial square root, mod n). No prime has such a square root.

Primality Testing

Proof: $n \in \text{PRIMES} \Rightarrow \Pr[\mathit{accepts}\ n] = 1$.
We just need to show that if the algorithm rejects, then *n* must be composite.

- Reject because of Fermat: Then not prime, by Fermat's Little Theorem (primes pass).
- Reject because of Carmichael: Then 1 has a nontrivial square root *b*, mod *n*, so *n* isn't prime, by Fact 2.

Proof: $n \notin \text{PRIMES} \Rightarrow \Pr[\mathit{accepts}\ n] \leq \frac{1}{2}$.

- Suppose *n* is a composite.
- If *n* is not a Carmichael number, then at least half of the possible choices of *a* fail the Fermat test (by Fact 1).
- If *n* is a Carmichael number, then Fact 2 says that some *b* fails the Carmichael test (is a nontrivial square root).
- Actually, when we generate *b* using *a* as above, at least half of the possible choices of *a* generate *b*s that fail the Carmichael test.

Primality Testing

So we have proved:

Theorem

1. $n \in Primes \implies Pr[\text{accepts } n] = 1$
2. $n \notin Primes \implies Pr[\text{accepts } n] \leq (\frac{1}{2})$

This implies:

Theorem

$PRIMES \in coRP$

Repeating k times, or using an amplification lemma, we get:

Theorem

1. $n \in Primes \implies Pr[\text{accepts } n] = 1$
2. $n \notin Primes \implies Pr[\text{accepts } n] \leq (\frac{1}{2})^k$

Thus, the algorithm might sometimes make mistakes and classify a composite as a prime, but the probability of doing this can be made arbitrarily low.