Randomized Computation

Siddhant Chaudhary

5 August 2021

1 Randomization: A brief introduction

Randomization is the technique of using outcomes of experiments in designing algorithms. It could be as simple as generating random numbers in a given range and using them in some particular fashion. Let us now see how complexity theorists deal with randomization.

Definition 1.1. Let $T : \mathbb{N} \to \mathbb{N}$ and let $L \subseteq \{0,1\}^*$. We say that a probabilistic Turing Machine M decides L in time $T(n)$, if for every $x \in \{0,1\}^*$, M halts in $T(|x|)$ steps regardless of its random choices, and

$$\mathbf{P}[M(x) = L(x)] \geq \frac{2}{3}$$

where we say $L(x) = 1$ if $x \in L$, and 0 otherwise. The set $\mathbf{BPTIME}(T(n))$ denotes the class of languages decided by probabilistic Turing Machines in $O(T(n))$ time, and we define

$$\mathbf{BPP} = \bigcup_c \mathbf{BPTIME}(n^c)$$

For this discussion, we don't need to dive deeply into proving properties of **BPP**; our main focus will be in understanding some of the general techniques used in randomization. However. the following fact is immediate, and will be left as an exercise for the reader.

Proposition 1.2. $\mathbf{P} \subseteq \mathbf{BPP}$.

Remark 1.3. If you're not familiar with complexity classes: **P** is the class of problems (technically, *languages*) solvable by deterministic polynomial time algorithms (try Googling these terms; the definitions are not very hard, but need the notion of Turing Machines). So all this proposition is stating is that any deterministic polynomial time algorithm you can think of is a randomized algorithm; one in which no randomization is being used at all.

Remark 1.4. The question whether the inclusion $\mathbf{BPP} \subseteq \mathbf{P}$ holds is still an open problem. Roughly, this question is asking whether every randomized algorithm can be converted to a deterministic polynomial time algorithm. Converting any randomized algorithm to a deterministic exponential time algorithm is possible; formally, $\mathbf{BPP} \subseteq \mathbf{EXP}$. Can you prove this?

2 Polynomial Identity Testing

2.1 Problem Description

Let F be a field. Most often, F is one of $\mathbb{Q}, \mathbb{R}, \mathbb{C}$, or it is a finite field. Let $f, g \in F[x_1, ..., x_n]$. We are interested in determining the equality

$$f = g$$

which is the same as checking whether

$$f - g = 0$$

is true or not. Ofcourse, the most naive way to check this is by checking whether all of the coefficients of $f - g$ are zero or not. However, as the following example, shows, this is not the best way to go complexity-wise.

Example 2.1. Suppose we want to verify something like

$$\prod_{i=1}^{6}(x-i) \overset{?}{=} x^6 - 7x^3 + 25$$

Here, expanding the product is not a good idea. In fact, as an exercise, try to show that the *number of multiplications* you need to do to expand the product grows exponentially as the number of factors increases.

2.2 A useful lemma

We will now state and prove the tool we will use to solve the PIT problem.

Theorem 2.2 (Schwartz-Zippel Lemma). *Let $p(x_1, ..., x_n)$ be any non-zero element of $F[x_1, ..., x_n]$ of degree d. Let $S \subseteq F$ be any finite set. If $a_1, ..., a_n$ are independently picked uniformly at random from S, then*

$$\mathbf{P}[p(a_1, ..., a_n) = 0] \leq \frac{d}{|S|}$$

Proof. We prove this by induction on the number of variables n. For the base case, suppose $n = 1$, i.e we are working in the ring $F[x]$. From PROMYS number theory, we know that any polynomial of degree d over a field F has atmost d roots in F. The base case is not obvious from this.

Now suppose the claim holds for all polynomials with atmost $n - 1$ variables. We regard p as a single variable polynomial with coefficients in $F[x_1, ..., x_{n-1}]$. Formally, we are using the isomorphism

$$F[x_1, ..., x_n] \cong F[x_1, ..., x_{n-1}][x_n]$$

(If you wish, you may prove this; the proof isn't very hard). So, we write

$$p(x_1, ..., x_n) = \sum_{i=0}^{d} x_n^i p_i(x_1, ..., x_{n-1})$$

where each $p_i \in F[x_1, ..., x_{n-1}]$. Since $p \neq 0$, there is a maximum index $j \leq d$ such that

$$p_j(x_1, ..., x_{n-1}) \neq 0$$

So, just to make the leading coefficient non-zero, we can write

$$p(x_1, ..., x_n) = \sum_{i=0}^{j} x_n^i p_i(x_1, ..., x_{n-1})$$

Since p has degree d, we note that

$$\deg p_k \leq d - k$$

for each $0 \leq k \leq j$. In particular, we have $\deg p_j \leq d - j$. Applying the induction hypothesis to p_j (which has $n-1$ variables), we see that

$$\mathbf{P}_{a_1,...,a_{n-1}\in S}[p_j(a_1,...,a_{n-1}) = 0] \leq \frac{d-j}{|S|}$$

We now *condition* the event $p(a_1,...,a_{n-1},a_n) = 0$ based on two *mutually exclusive and exhaustive events* (see definitions of this if you don't know what this means. Also, see **Exercise 2.4** below).

1. In the first case, $p(a_1,...,a_{n-1},a_n) = 0$ and $p_j(a_1,...,a_{n-1}) = 0$. By the trivial bound,

$$\begin{aligned}\mathbf{P}[p(a_1,...,a_{n-1}) = 0 \wedge p_j(a_1,...,a_{n-1}) = 0] &\leq \mathbf{P}[p_j(a_1,...,a_{n-1}) = 0] \\ &\leq \frac{d-j}{|S|}\end{aligned}$$

2. In the second case, $p(a_1,...,a_{n-1},a_n) = 0$ and $p_j(a_1,...,a_{n-1}) \neq 0$. For fixed $a_1,...,a_{n-1} \in F$, consider the one variable polynomial

$$g(x) = p(a_1,...,a_{n-1},a_n) = \sum_{i=0}^{j} x^i p_i(a_1,...,a_{n-1})$$

Clearly, $g \in F[x]$, and g is non-zero. So by the one variable bound (and keeping in mind that $a_1,...,a_{n-1}$ are fixed), we see that

$$\mathbf{P}_{a_n\in S}[g(a_n) = 0] \leq \frac{j}{|S|}$$

Now, conditioning on the possible values of $a_1,...,a_{n-1}$ (see **Exercise 2.4** below), we get that

$$\mathbf{P}[p(a_1,...,a_{n-1}) = 0 \wedge p_j(a_1,...,a_{n-1}) \neq 0] \leq \frac{j}{|S|}$$

Summing the probabilities of the two disjoint events above, we see that

$$\mathbf{P}[p(a_1,...,a_n) = 0] \leq \frac{d-j}{|S|} + \frac{j}{|S|} = \frac{d}{|S|}$$

and this completes the induction proof. □

Exercise 2.3. For events X, Y, define the *conditional probability* $\mathbf{P}(X|Y)$ as

$$\mathbf{P}(X|Y) = \frac{\mathbf{P}(X \cap Y)}{\mathbf{P}(Y)}$$

Intuitively, this is the probability of the event X *provided* event Y has occurred. Suppose $E_1,...,E_n$ are mutually exclusive and exhaustive events for a sample space. Let X be any event. Show that

$$\mathbf{P}(X) = \sum_{i=1}^{n} \mathbf{P}(X|E_i)\mathbf{P}(E_i) = \sum_{i=1}^{n} \mathbf{P}(X \cap E_i)$$

Exercise 2.4. Suppose $E_1, ..., E_n$ are mutually exclusive and exhaustive events. Let X be any event such that

$$\mathbf{P}(X|E_1) = \mathbf{P}(X|E_2) = \cdots = \mathbf{P}(X|E_n)$$

Show that

$$\mathbf{P}(X) = \mathbf{P}(X|E_1) = \mathbf{P}(X|E_2) = \cdots = \mathbf{P}(X|E_n)$$

So, to compute $\mathbf{P}(X)$, we can just assume that event E_i has occurred, for any $1 \le i \le n$. Can you now see how a variant of this fact has been used in the proof above? In the proof, we are working in inequalities rather than equalities, but the basic idea is the same.

2.3 A randomized algorithm for PIT

We now give a randomized algorithm for PIT using the lemma we proved above. The algorithm is as follows.

1. Suppose the input polynomials are $f, g \in F[x_1, ..., x_n]$. We need to check

$$f - g = 0$$

2. Let $d = \deg(f - g)$. Take $S \subseteq F$ such that $|S| = 100d$.
3. Pick a vector $(a_1, ..., a_n) \in S^n$ uniformly at random, and check the equality

$$(f - g)(a_1, ..., a_n) = 0$$

4. If the equality holds, return **true**; else return **false**.

Exercise 2.5. Prove that this algorithm has a *one-sided* error with an upper bound of 1/100, which is good enough for us. So we now have our algorithm!

Remark 2.6. A *one-sided error* means that the algorithm works with no error in all cases which are actually true, and work with some error in cases which are false.

3 Applying PIT to Bipartite Matching

3.1 Perfect Matchings

In this section, I'm assuming you are familiar with graphs; if not, just Google the definitions. They're not hard. I'm also assuming you are familiar with *bipartite graphs*. Look at their definitions too if you need to.

Definition 3.1. Let $G = (V, E)$ be a graph. A *matching* is a subset $M \subseteq E$ of edges such that the following holds: for each vertex $x \in V$, there is atmost one edge $e \in M$ incident on x. A *perfect matching* is a matching in which each vertex $x \in V$ is incident to atleast one edge $e \in M$. In this section, we will be interested in perfect matchings of bipartite graphs.

Having these definitions in mind, consider the following scenario: suppose $G = (V, E)$ is a bipartite graph. Also, suppose the bipartition is $V_1 \cup V_2$, with $|V_1| = |V_2| = n$. Let A_G be the *symbolic adjacency matrix* of the graph. A_G is defined as follows.

$$A_G[ij] = \begin{cases} x_{ij} & , \text{ if } i \in V_1 \text{ and } j \in V_2 \text{ are connected} \\ 0 & , \text{ otherwise} \end{cases}$$

Note that here x_{ij} are formal symbols, and they should be treated as so. The polynomial in which we'll be interested is $\det A_G$, i.e the determinant of A_G. Clearly, $\det A_G$ is a polynomial in atmost n^2 variables. Also, we'll be using the *permutation expansion of the determinant*:

$$\det A_G = \sum_{\sigma \in S_n} \epsilon(\sigma) A_{1\sigma(1)} \cdots A_{n\sigma(n)}$$

(If you aren't familiar with this, just read this on Wikipedia. Knowing where this formula comes from is not important for now). Here S_n is the group of permutations of n numbers, and $\epsilon(\sigma)$ is the *sign* of the permutation σ.

Proposition 3.2. *The graph G above has a perfect matching if and only if* $\det A_G \neq 0$. *Here* $\det A_G$ *is a polynomial, and so we are working with polynomial equalities.*

Exercise 3.3. Prove the above claim. See how perfect matchings and permutations are related here. This is not very difficult; this exercise only checks whether you've understood the problem statement, and whether you're comfortable with polynomials.

3.2 The Algorithm

Using all of our observations, we now have the following algorithm at our disposal.

1. Suppose our input graph is $G = (V, E)$. As before, our assumption is that $V = V_1 \cup V_2$ is a bipartition with $|V_1| = |V_2| = n$.
2. Compute A_G, and $\det A_G$. Use standard algorithms to compute the determinant (Note: here we are compuing $\det A_G$ as a polynomial).
3. Use PIT with $F = \mathbb{Q}$ and check whether $\det A_G = 0$. If equality holds, return *false*; else return *true*.

Exercise 3.4. Show that this algorithm has a one-sided error. Use **Proposition 3.2**.

4 Verifying Matrix Multiplication

4.1 Problem Statement

Suppose F is our base field. Throughout this discussion, we assume $F = \mathbb{F}_2$, although any finite field will be fine. We are given $n \times n$ matrices A, B and C over F, and we want to check whether the equality

$$AB = C$$

holds. If we use the usual matrix multiplication algorithm, we will need $\Theta(n^3)$ time; although this is not bad, we can do better with randomization.

4.2 The Algorithm

Here is the algorithm that we'll use.

1. Pick any vector $r = (r_1, ..., r_n) \in F^n$ uniformly at random.

2. Check the equality
$$(AB)r = Cr$$
3. If the equality holds, return **true**; else return **false**.

If you wish, you can analyze this algorithm through the following exercises.

Proposition 4.1. *If $AB \neq C$ and $F = \mathbb{F}_2$, then*
$$\mathbf{P}[ABr = Cr] \leq \frac{1}{2}$$
where the probability is being taken over the choice of $r \in F^n$.

Exercise 4.2. Let M be any non-zero matrix. Show that if $r \in F^n$ is chosen uniformly at random, then
$$\mathbf{P}[Mr = 0] \leq \frac{1}{2}$$
where again $F = \mathbb{F}_2$. Hint: since M is non-zero, it has a non-zero entry, say M_{11} (without loss of generality. Question: why can we assume this without loss of generality?). Now,
$$Mr = 0 \implies \sum_{j=1}^{n} M_{1j} r_j = 0$$
Rewrite the last equation, by separating out r_1. Using something similar to **Exercise 2.4**, condition on the values of $r_2, ..., r_n$. Get the bound.

Exercise 4.3. Prove **Proposition 4.1.**

Exercise 4.4. Show that the above algorithm has one-sided error. In **Exercise 4.2**, we've shown that the error bound is $1/2$, which is not great. Show that, by repeating the algorithm t times, we can reduce the error bound to $(1/2)^t$. Can we do something similar for two-sided errors?

5 Other Applications

Randomization is a powerful technique. Other notable applications are given below.

1. Randomization is used in **universal hashing**. *Hashing* in general is a technique used to store and query data. There are certain families of hash functions, called *universal hash families*, which are useful because they reduce the collision probability. Moreover, such a family can be applied to any set of keys.
2. Many primality tests use randomization. The most notable one is the *Miller-Rabin Primality Test.* If you wish, you can read about it on the internet.