Randomized Computation

Siddhant Chaudhary

PROMYS 2021

Flip Coins!

1. **Randomization** is the technique of using outcomes of experiments in designing useful algorithms.

Flip Coins!

1. **Randomization** is the technique of using outcomes of experiments in designing useful algorithms.
2. An example could be the generation of random numbers (technically, **pseudorandom number generation**) in an algorithm.

Flip Coins!

1. **Randomization** is the technique of using outcomes of experiments in designing useful algorithms.
2. An example could be the generation of random numbers (technically, **pseudorandom number generation**) in an algorithm.
3. Mathematically defined as languages recognized by **Probabilistic Turing Machines** with small error bound. The class of languages is denoted **BPP** (Trivially $\mathbf{P} \subseteq \mathbf{BPP}$. Converse is an open problem).

Our first randomized algorithm - PIT

- You are working in the ring $F[x_1, ..., x_n]$, where F is a field. Often $F = \mathbb{Q}$.

Our first randomized algorithm - PIT

- You are working in the ring $F[x_1, ..., x_n]$, where F is a field. Often $F = \mathbb{Q}$.
- $f(x_1, ..., x_n), g(x_1, ..., x_n) \in F[x_1, ..., x_n]$.

Our first randomized algorithm - PIT

- You are working in the ring $F[x_1, ..., x_n]$, where F is a field. Often $F = \mathbb{Q}$.
- $f(x_1, ..., x_n), g(x_1, ..., x_n) \in F[x_1, ..., x_n]$.
- Need to determine whether

$$f = g$$

which is the same as determining whether

$$f - g = 0$$

Our first randomized algorithm - PIT

Example

In some scenarios, either f or g might be given in terms of linear factors. For instance, we might need to verify the following.

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

Expanding the product is not a good idea! If the 6 is replaced by a large constant, this becomes difficult.

A Useful Tool

Proposition

(Schwartz-Zippel) 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 picked uniformly at random from S, then

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

The Proof

- By induction on the number of variables n ; the base case of one variable is easy.

The Proof

- By induction on the number of variables n ; the base case of one variable is easy.
- Assume the claim holds for all polynomials with atmost $n-1$ variables.

The Proof

- By induction on the number of variables n ; the base case of one variable is easy.
- Assume the claim holds for all polynomials with atmost $n - 1$ variables.
- Regard p as a single variable polynomial with coefficients in $F[x_1, ..., x_{n-1}]$. Formally, we are using

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

The Proof

- So we write

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

The Proof

- So we write

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

- Since $p \neq 0$, there is a maximum index $j \leq d$ such that $p_j(x_1, ..., x_{n-1}) \neq 0$. So, we can write

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

The Proof

- 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 p_j, we see that

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

The Proof

- We condition the event $p(a_1, ..., a_{n-1}, a_n) = 0$ based on two mutually exclusive and exhaustive events.

The Proof

- We condition the event $p(a_1, ..., a_{n-1}, a_n) = 0$ based on two mutually exclusive and exhaustive events.
    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,

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

The Proof

- We condition the event $p(a_1, ..., a_{n-1}, a_n) = 0$ based on two mutually exclusive and exhaustive events.
  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,
  $$\begin{aligned} &\mathbf{P}[p(a_1, ..., a_{n-1}, a_n) = 0 \wedge p_j(a_1, ..., a_{n-1}) = 0] \\ &\leq \mathbf{P}[p_j(a_1, ..., a_{n-1}) = 0] \\ &\leq \frac{d-j}{|S|} \end{aligned}$$
  2. In the second case, $p(a_1, ..., a_{n-1}, a_n) = 0$ and $p_j(a_1, ..., a_{n-1}) \neq 0$. Consider the one variable polynomial
  $$g(x) = p(a_1, ..., a_{n-1}, x) = \sum_{i=0}^{j} x^i p_i(a_1, ..., a_{n-1})$$

The Proof

- By the one variable bound, we have

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

The Proof

- By the one variable bound, we have

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

- Summing the two probabilities above, we get

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

Application to PIT

- Suppose our input polynomials are $f, g \in F[x_1, ..., x_n]$; we need to determine whether

$$f - g = 0$$

Application to PIT

- Suppose our input polynomials are $f, g \in F[x_1, ..., x_n]$; we need to determine whether

$$f - g = 0$$

- Take $S \subseteq F$ such that $|S| = 100d$.

Application to PIT

- Suppose our input polynomials are $f, g \in F[x_1, ..., x_n]$; we need to determine whether

$$f - g = 0$$

- Take $S \subseteq F$ such that $|S| = 100d$.
- Pick a random vector $(a_1, ..., a_n) \in S^n$, and check the equality

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

  If the equality is true, return TRUE; otherwise return FALSE.

Application to PIT

- Suppose our input polynomials are $f, g \in F[x_1, ..., x_n]$; we need to determine whether

$$f - g = 0$$

- Take $S \subseteq F$ such that $|S| = 100d$.
- Pick a random vector $(a_1, ..., a_n) \in S^n$, and check the equality

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

  If the equality is true, return TRUE; otherwise return FALSE.
- By **Schwarz-Zippel**, we have a one-sided error bound of $\frac{d}{100d} = \frac{1}{100}$, which is very small!

Applying PIT to Bipartite Matching

- **Bipartite Graph**: two teams, 1V1 matches within the two teams.

Applying PIT to Bipartite Matching

- **Bipartite Graph**: two teams, 1V1 matches within the two teams.
- We are interested in **perfect matchings**. This means that a player from one team can play against *atmost one* player from the second team, and *every* player from either team has to play.

Applying PIT to Bipartite Matching

- **Bipartite Graph**: two teams, 1V1 matches within the two teams.
- We are interested in **perfect matchings**. This means that a player from one team can play against *atmost one* player from the second team, and *every* player from either team has to play.

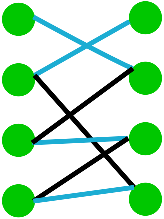

Figure: Edges in cyan form a perfect matching.

Applying PIT to Bipartite Matching

- Suppose the bipartition is $V_1 \cup V_2$ with $|V_1| = |V_2| = n$.

Applying PIT to Bipartite Matching

- Suppose the bipartition is $V_1 \cup V_2$ with $|V_1| = |V_2| = n$.
- Let A_G be the **symbolic adjacency matrix**, defined as follows.

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

Applying PIT to Bipartite Matching

- Suppose the bipartition is $V_1 \cup V_2$ with $|V_1| = |V_2| = n$.
- Let A_G be the **symbolic adjacency matrix**, defined as follows.

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

- In the graph in the previous slide, A_G is the following matrix.

$$A_G = \begin{bmatrix} 0 & x_{12} & 0 & 0 \\ x_{21} & 0 & 0 & x_{24} \\ 0 & x_{32} & x_{33} & 0 \\ 0 & 0 & x_{43} & x_{44} \end{bmatrix}$$

Applying PIT to Bipartite Matching

- Our polynomial will be $\det A_G$, a polynomial in n^2 variables.

Applying PIT to Bipartite Matching

- Our polynomial will be $\det A_G$, a polynomial in n^2 variables.
- Use the permutation expansion of the determinant.

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

Applying PIT to Bipartite Matching

- Our polynomial will be $\det A_G$, a polynomial in n^2 variables.
- Use the permutation expansion of the determinant.

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

- Claim: G has a perfect matching if and only if

$$\det A_G \neq 0$$

Applying PIT to Bipartite Matching

- Our polynomial will be $\det A_G$, a polynomial in n^2 variables.
- Use the permutation expansion of the determinant.

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

- Claim: G has a perfect matching if and only if

$$\det A_G \neq 0$$

- Use PIT with $F = \mathbb{Q}$ to get a randomized algorithm.

Verifying Matrix Multiplication

- F is our base field (for simplicity, let $F = \mathbb{F}_2$). We are given matrices A, B and C of dimension $n \times n$.

Verifying Matrix Multiplication

- F is our base field (for simplicity, let $F = \mathbb{F}_2$). We are given matrices A, B and C of dimension $n \times n$.
- We want to verify the equation

$$AB = C$$

If we use the usual matrix-multiplication algorithm, the time taken is $\Theta(n^3)$.

Verifying Matrix Multiplication

- F is our base field (for simplicity, let $F = \mathbb{F}_2$). We are given matrices A, B and C of dimension $n \times n$.
- We want to verify the equation

$$AB = C$$

  If we use the usual matrix-multiplication algorithm, the time taken is $\Theta(n^3)$.
- Instead, we use a randomized approach; pick a vector $\boldsymbol{r} = (r_1, ..., r_n) \in F^n$ uniformly at random. Check the equality

$$(AB)\boldsymbol{r} = C\boldsymbol{r}$$

  If the equality holds, return TRUE; else return FALSE.

Verifying Matrix Multiplication

- Claim: if $AB \neq C$ and $F = \mathbb{F}_2$, then

$$\mathbf{P}[AB\boldsymbol{r} = C\boldsymbol{r}] \leq \frac{1}{2}$$

Verifying Matrix Multiplication

- Claim: if $AB \neq C$ and $F = \mathbb{F}_2$, then

$$\mathbf{P}[AB\boldsymbol{r} = C\boldsymbol{r}] \leq \frac{1}{2}$$

- If M is any non-zero $n \times n$ matrix, then it has some non-zero entry, say M_{11}. Also,

$$M\boldsymbol{r} = 0 \implies \sum_{j=1}^{n} M_{1j} r_j = 0$$

which means

$$r_1 = \frac{-\sum_{j=2}^{n} M_{1j} r_j}{M_{11}}$$

Verifying Matrix Multiplication

- Now condition on the values $(r_2, ..., r_n)$, i.e fix $(r_2, ..., r_n)$. Verify now that the last equation holds with probability less than $\dfrac{1}{2}$.

Verifying Matrix Multiplication

- Now condition on the values $(r_2, ..., r_n)$, i.e fix $(r_2, ..., r_n)$. Verify now that the last equation holds with probability less than $\frac{1}{2}$.
- Summing up all the conditional probabilities, we obtain

$$\mathbf{P}[AB\boldsymbol{r} = C\boldsymbol{r}] \leq \frac{1}{2}$$

Verifying Matrix Multiplication

- Now condition on the values $(r_2, ..., r_n)$, i.e fix $(r_2, ..., r_n)$. Verify now that the last equation holds with probability less than $\frac{1}{2}$.
- Summing up all the conditional probabilities, we obtain

$$\mathbf{P}[AB\boldsymbol{r} = C\boldsymbol{r}] \leq \frac{1}{2}$$

- So our algorithm has a one-sided error less than $\frac{1}{2}$; still not very nice. How to fix this?

Verifying Matrix Multiplication

- Now condition on the values $(r_2, ..., r_n)$, i.e fix $(r_2, ..., r_n)$. Verify now that the last equation holds with probability less than $\dfrac{1}{2}$.
- Summing up all the conditional probabilities, we obtain

$$\mathbf{P}[AB\boldsymbol{r} = C\boldsymbol{r}] \leq \frac{1}{2}$$

- So our algorithm has a one-sided error less than $\frac{1}{2}$; still not very nice. How to fix this?
- Repeat the algorithm t times independently, to make the error probability less than $\left(\frac{1}{2}\right)^t$. $t = 100$ will give a good enough bound.

Randomization is powerful

- Used in **universal and perfect hashing**, a powerful technique for storing and querying large amounts of data in average $O(1)$ time.

Randomization is powerful

- Used in **universal and perfect hashing**, a powerful technique for storing and querying large amounts of data in average $O(1)$ time.
- Primality Tests, like *Miller-Rabin*.

Randomization is powerful

- Used in **universal and perfect hashing**, a powerful technique for storing and querying large amounts of data in average $O(1)$ time.
- Primality Tests, like *Miller-Rabin*.
- Get as accurate as you want! One-sided errors can be made as small as possible, by introducing a parameter. This is just the idea of independence of events.

Randomization is powerful

- Used in **universal and perfect hashing**, a powerful technique for storing and querying large amounts of data in average $O(1)$ time.
- Primality Tests, like *Miller-Rabin*.
- Get as accurate as you want! One-sided errors can be made as small as possible, by introducing a parameter. This is just the idea of independence of events.
- Hope you enjoyed the discussion!