Randomized Computation

- Deterministic and (Las Vegas & Monte Carlo) Randomized Algorithms
- Probability Review
- Probabilistic Analysis of deterministic QUICK-SORT Algorithm
- RANDOMIZED-SELECT and RANDOMIZED-QUICK-SORT
- Max-Cut
- Min-Cut
- MAX-3-SAT and Derandomization
- Closest Pair
- Randomized Complexity Classes

Imdad ullah Khan

Randomized Complexity Classes

Traditional complexity classes (e.g., P, NP) primarily consider deterministic computations

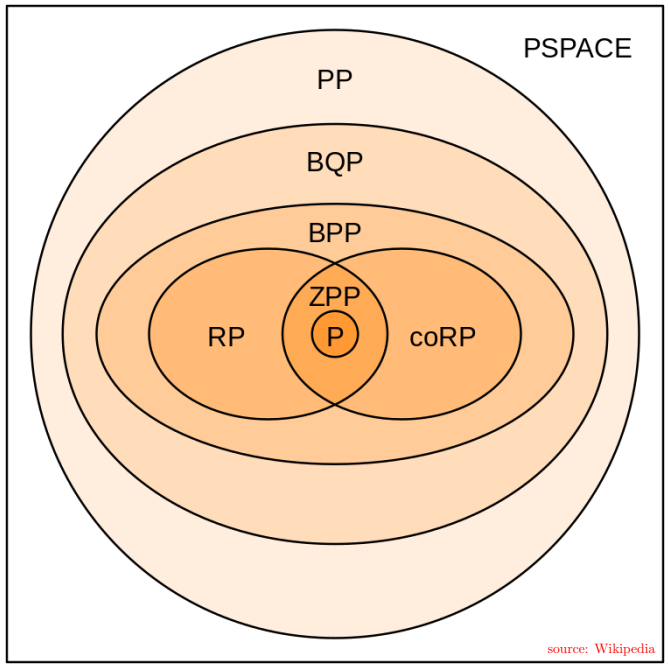
Randomized Complexity Classes introduce *randomness* into the decision process offering unique insights into how algorithms perform with an element of chance

Goals:

- To understand the **power and limitations** of randomized algorithms
- To explore the **hierarchy** of complexity classes and how randomness affects computational resources
- To evaluate the **practical applications** of such algorithms in real-world scenarios

Randomized Complexity Classes

- RP (Random Polynomial Time): Problems solvable by algorithms that may err only when answering **Yes**
- co-RP: Complementary to RP; algorithms may err only when answering **No**
- ZPP (Zero-error Probabilistic Polynomial Time): Problems with algorithms always returning correct answer in random time
- BPP (Bounded-error Probabilistic Polynomial Time): Problems with polynomial time algorithms returning probably correct answer



Significance in Theoretical CS and Applications

- Understanding these classes sheds light on the *inherent randomness* in computational processes
- Influences areas like cryptography, computational biology, and machine learning
- Challenges the boundary between *what is computable* deterministically and probabilistically

Class RP (Random Polynomial Time)

A decision problem is in RP for which there exists a polynomial-time randomized algorithm such that:

- If the correct answer is **No**, the algorithm always returns **No**
- If the correct answer is **Yes**, the algorithm returns **Yes** with a probability greater than $1/2$

RP algorithms exhibit *one-sided error* for **Yes** instances

- Offers a **probabilistic approach** to decision problems, where determinism may not provide efficient solutions
- Provides a foundation for understanding how randomness can be harnessed to improve computational efficiency
- Serves as a stepping stone to more complex randomized complexity classes like BPP and ZPP

Primality Testing: An RP Example

Input: A positive integer n

Output: **Yes**, if n is prime, else **No**

- Cryptography: Essential for generating keys in cryptographic algorithms like RSA
- Random Number Generation: Used in creating seeds for pseudorandom number generators
- Error Checking: Employed in algorithms for error detection and correction

Fermat's Little Theorem: (basis for many primality tests)

If n is a prime number, then for any integer a, $a^{n-1} \equiv 1 \pmod{n}$

The Miller-Rabin Primality Test

A widely used probabilistic test that relies on modular exponentiation

Algorithm Miller-Rabin Primality Test (simplified)

procedure IsPrime(n, k)
  for $i \leftarrow 1$ to k **do**
    Choose a randomly in the range $[2, n-2]$
    if $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
      return No ▷ n is composite
  return Yes ▷ n is probably prime

- **Yes** answer might be wrong; **No** answer is always right
- Runs k rounds of tests, each passed test reduce the error probability
- Demonstrates the **amplification** technique in RP

The Relationship Between RP and P

$$P \subseteq RP$$

▷ If a problem is in P, it is also in RP (since deterministic algorithms are a special case of randomized algorithms)

The big question: Are there problems in RP that are not in P?

This question explores the **boundaries of efficiency** for deterministic vs. randomized algorithms

Class co-RP: The Complement of RP

A decision problem is in co-RP if its complement is in RP. That is, there exists a polynomial-time randomized algorithm such that:

- If the correct answer is **Yes**, the algorithm always returns **Yes**
- If the correct answer is **No**, the algorithm returns **No** with a probability greater than $1/2$

RP algorithms exhibit *one-sided error* for **No** instances

source: Wikipedia

RP algorithm (1 run)		
Correct answer \ Answer produced	Yes	No
Yes	≥ 1/2	≤ 1/2
No	0	1

source: Wikipedia

co-RP algorithm (1 run)		
Correct answer \ Answer produced	Yes	No
Yes	1	0
No	≤ 1/2	≥ 1/2

Polynomial Identity Testing: A co-RP Example

Input: Two polynomials P and Q

Output: **Yes**, if they are identical for all variable assignments, else **No**

- Computer Algebra Systems (symbolic computation): Crucial for simplifying expressions and verifying algebraic identities
- Coding Theory: Used in error-detecting and error-correcting codes
- Complexity Theory: Helps in proving lower bounds for arithmetic circuits

The problem is trivial if we can read P and Q directly in poly-time;

We only have black box access, i.e., we get $P(\mathbf{x})$ for given input x

Polynomial Identity Testing: Deterministic Algorithm

Univariate case:

Let d be the degree of P and Q and suppose we are computing in a field $\mathcal{F}$,

$\implies$ P and Q have d roots each

Pick $d+1$ distinct values at random from $\mathcal{F}$, if $P(x) - Q(x) = 0$ for all these points, then P and Q are identical

Multivariate case:

This approach does not directly apply to the multivariate case because there can be exponentially many roots

Can handle multivariate case by fixing $n-1$ variables and applying the result from the univariate case

Schwartz–Zippel Lemma for Polynomial Identity Testing

Assume that we have some subset $S \in \mathcal{F}$ with $|S| \geq 2d$

Algorithm Polynomial Identity Testing using Schwartz–Zippel

procedure AREPOLYNOMIALSEQUAL(P, Q, n, k)
  for $i \leftarrow 1$ to k **do**
    Choose a random assignment $\mathbf{r_1}, ..., \mathbf{r_n} \in \mathbf{R^d}$
    if $P(\mathbf{r_1}, ..., \mathbf{r_n}) \neq Q(\mathbf{r_1}, ..., \mathbf{r_n})$ **then**
      return No ▷ Polynomials are not identical
  return Yes ▷ Polynomials are probably identical

- $P(x) = x^2 + 2x + 1$ and $Q(x) = (x + 1)^2$, with $k = 5$ and random choices showing they are identical
- $P(x) = x^3 + x$ and $Q(x) = x^3 + 2x$, with $k = 5$ and one random choice showing they are not identical

$$Pr[P(r_1, ..., r_n) - Q(r_1, ..., r_n) = 0] \leq {}^{d}/_{|S|}$$

Relation of co-RP to RP and Other Classes

- co-RP complements RP by offering probabilistic guarantees for the opposite type of answer
- Understanding both RP and co-RP is crucial for **comprehensive insights** into how randomness affects computational complexity
- The intersection of RP and co-RP forms the class ZPP, indicating problems solvable with **zero error probability** using probabilistic methods

Class ZPP: Zero-error Probabilistic Polynomial Time

A decision problem is in RP for which there exists a randomized algorithm such that:

- the algorithm always returns the correct answer
- the expected runtime of the algorithm is polynomial (in input size)

Unlike RP or co-RP, ZPP algorithms **never err**, but their runtime is probabilistic

ZPP algorithms guarantee **absolute correctness**—they either provide the right answer or run indefinitely without producing an incorrect result

ZPP: Intersection of RP and co-RP

ZPP = RP ∩ co-RP

$\Longrightarrow$ If a problem X is in RP ∩ co-RP, then it has a ***Las Vegas*** algorithm

Let A and B be the algorithms making X in RP and co-RP, resp.

▷ Note that A and B can be completely different algorithms

Given an instance I of X,

1. Run A on I **for one step**. If it returns **yes**, the answer must be **Yes**
2. else, run B on I for one step. If it returns **No**, the answer must be **No**
3. If neither of the above occurs, repeat step 1 and 2

- Note only of A and B can give the wrong answer. Probability of that algorithm giving the wrong answer during each iteration is at most $1/2$
- The probability of doing k rounds is at most $1/2^k$
- Thus, the expected running time is polynomial

This shows that RP ∩ co-RP ⊆ ZPP

ZPP: Intersection of RP and co-RP

$\text{ZPP} = \text{RP} \cap \text{co-RP}$

$\Longleftarrow$ If a problem X has a ***Las Vegas*** algorithm, then it is in RP $\cap$ co-RP

Let C be a Las Vegas algorithm for X

Construct the following RP algorithm as follow: Given an instance I of X,

1. Run C on I **for** $\geq$ **double its expected runtime**. Returns its answer
2. If it does not give an answer return **No**

By Markov's Inequality, the probability that C yields an answer before we stop it is $\geq 1/2$

The probability we give wrong answer on a **Yes** instance is at most $1/2$

Thus, by definition, $X \in \text{RP}$

Similar, argument proves that $X \in$ co-RP

This shows that $\text{ZPP} \subseteq \text{RP} \cap \text{co-RP}$

Finding a Nash Equilibrium: A ZPP Challenge

Input: n players each with a set of strategies

Output: A strategy configuration where no player can benefit by changing strategies while the others remain constant ▷ Nash Equilibrium

Nash Equilibria are central to **game theory** and have implications in economics, computer science, and beyond.

- **Economics:** Predicting stable outcomes in competitive markets
- **Computer Science:** Designing efficient and reliable algorithms for networking, cryptography, and algorithmic game theory
- **Biology:** Understanding evolutionary stable strategies

Lemke-Howson Algorithm for Nash Equilibria

- Finds one Nash Equilibrium for two-player games.
- It systematically explores the game's strategy space using a pivoting method similar to those used in linear programming.
- While not strictly a ZPP algorithm, its deterministic nature and polynomial-time performance in practice align with ZPP's philosophy.

Algorithm Lemke-Howson Algorithm for Nash Equilibria

1: Initialize strategy profile S
2: **while** not converged to Nash Equilibrium **do**
3: Select a starting strategy for players
4: Perform best response dynamics
5: Adjust strategies based on other players' responses
6: **return** S as Nash Equilibrium

- For a two-player game with payoff matrices leading to a pure strategy Nash Equilibrium. Iterations adjust strategies until convergence.
- For a game where mixed strategies form a Nash Equilibrium, illustrating the algorithm's adjustment of probability distributions over strategies.

The Class BPP: Bounded-error Probabilistic Polynomial Time

A decision problem is in BPP for which there exists a polynomial-time randomized algorithm such that:

- the algorithm returns correct answer with a probability $\geq 2/3$

BPP algorithms allow for a *bounded-error probability*

For any given instance, the probability that a BPP algorithm returns the incorrect answer is less than or equal to *nicefrac*13

Amplification: By running a BPP algorithm multiple times and taking a majority vote, the error probability can be reduced exponentially, making it negligibly small

BPP Compared to RP and ZPP

RP vs. BPP: RP allows for one-sided error for **Yes** instances, while BPP allows errors on both **Yes** and **No** instances but with bounded probabilities

ZPP vs. BPP: ZPP ensures zero-error but with potentially unbounded runtime, whereas BPP has a controllable error rate with polynomial time

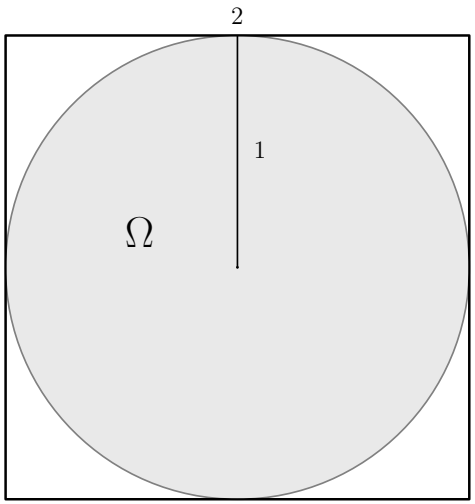
- Practical Implications: BPP captures the class of problems for which efficient and reliable randomized algorithms can be devised
- Theoretical Interest: The exact relationship between BPP and deterministic classes like P and NP is an open question in complexity

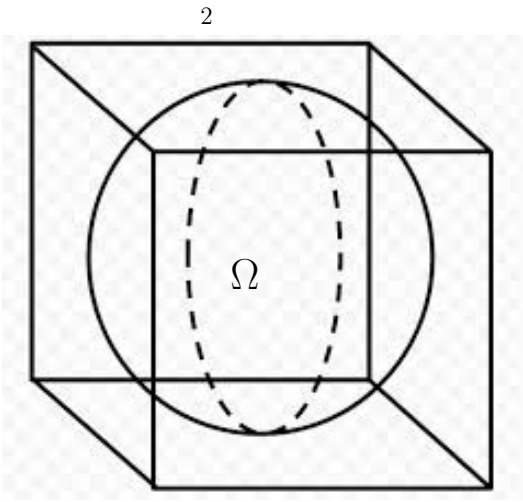
- Cryptography: For generating and verifying cryptographic keys and protocols under uncertainty
- Quantum Computing: BPP is also considered in the context of quantum algorithms, where probabilistic outcomes are natural
- Scientific Computing: In simulations and models where exact calculations are infeasible, BPP offers a way with bounded error

Estimating Volume of complex Geometric Objects

Estimate the volume of an irregular object for which the volume cannot be calculated directly through mathematical formulas

Crucial in material science, architecture, and computational geometry to determine material properties, construction planning, and spatial analysis





The bounding box represents a known volume within which the object (Ω) is contained.

Monte Carlo Algorithm for Estimating Volume

Algorithm Estimate Geometric Volume

```
1: procedure EstimateVolume(Object Ω, N)
2:     inside ← 0
3:     for i ← 1 to N do
4:         Generate a random point p within bounding box
5:         if p is inside Ω then                    ▷ this step may not be simple
6:             inside ← inside + 1
7:     return inside/N × Volume of bounding box
```

The ratio of points inside Ω to the total number of points approximates the object's volume relative to the bounding box.

It's a versatile method applicable to various geometrical shapes and sizes (as long as we can verify a point being inside)

Monte Carlo Example: Estimating π

Problem: Estimate the value of π by simulating random points within a unit circle enclosed in a 2×2 square

Algorithm Estimate Pi using Monte Carlo

procedure EstimatePi(N)
  $inside \leftarrow 0$
  for $i \leftarrow 1$ to N **do**
    $x, y \leftarrow$ random numbers between -1 and 1
    if $x^2 + y^2 \leq 1$ **then** ▷ If point (x, y) is within the unit circle centered at 0
      $inside \leftarrow inside + 1$
  return $inside/N \times 4$