# Design and Analysis of Algorithms
# CS218M
## Randomized Algorithms

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

# Randomized Algorithms

- Different from Deterministic Algorithms

# Randomized Algorithms

- Different from Deterministic Algorithms
- Different from Non-deterministic Algorithms (aka NP problems)

# Randomized Algorithms

- Different from Deterministic Algorithms
- Different from Non-deterministic Algorithms (aka NP problems)
- As atomic steps, the algorithm uses a random number generator with specified distribution over specified set outcomes.

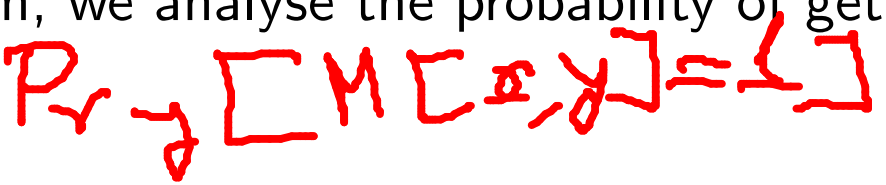
# Randomized Algorithms

- Different from Deterministic Algorithms
- Different from Non-deterministic Algorithms (aka NP problems)
- As atomic steps, the algorithm uses a random number generator with specified distribution over specified set outcomes.
- Result of the algorithm depends on input as well as the random number generated.

$$P_y[M[x,y]=1]$$

# Randomized Algorithms

- Different from Deterministic Algorithms
- Different from Non-deterministic Algorithms (aka NP problems)
- As atomic steps, the algorithm uses a random number generator with specified distribution over specified set outcomes.
- Result of the algorithm depends on input as well as the random number generated.
- For a decision problem, we analyse the probability of getting correct answer.



# Randomized Algorithms

- Different from Deterministic Algorithms
- Different from Non-deterministic Algorithms (aka NP problems)
- As atomic steps, the algorithm uses a random number generator with specified distribution over specified set outcomes.
- Result of the algorithm depends on input as well as the random number generated.
- For a decision problem, we analyse the probability of getting correct answer.
- With repeated trials we can guarantee success with almost 1 probability. We analyse expected running time.

# Approximate MaxSAT

Optimization Problem Given a rectified $3CNF$ formula $\phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, the aim is to find an assignment which satisfies the maximum number of clauses. Value of solution is number of clauses satisfied. We approximate MaxSAT.

# Approximate MaxSAT

Optimization Problem Given a rectified $3CNF$ formula $\phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, the aim is to find an assignment which satisfies the maximum number of clauses. Value of solution is number of clauses satisfied. We approximate MaxSAT.

- An assignment is uniformly randomly generated.

# Approximate MaxSAT

Optimization Problem Given a rectified $3CNF$ formula $\phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, the aim is to find an assignment which satisfies the maximum number of clauses. Value of solution is number of clauses satisfied. We approximate MaxSAT.

- An assignment is uniformly randomly generated.
- Random variable $Z$ gives the number of satisfied clauses.

# Approximate MaxSAT

Optimization Problem Given a rectified $3CNF$ formula $\phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, the aim is to find an assignment which satisfies the maximum number of clauses. Value of solution is number of clauses satisfied. We approximate MaxSAT.

- An assignment is uniformly randomly generated.
- Random variable $Z$ gives the number of satisfied clauses.
- Indicator random variable $Z_i = 1$ iff clause $C_i$ is satisfied. Hence $Z = \Sigma_{i=1}^{k} Z_i$.

# Approximate MaxSAT

Optimization Problem Given a rectified $3CNF$ formula $\phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, the aim is to find an assignment which satisfies the maximum number of clauses. Value of solution is number of clauses satisfied. We approximate MaxSAT.

- An assignment is uniformly randomly generated.
- Random variable $Z$ gives the number of satisfied clauses.
- Indicator random variable $Z_i = 1$ iff clause $C_i$ is satisfied. Hence $Z = \Sigma_{i=1}^{k} Z_i$.
- In rectified $3CNF$, each clause $C_i = (l_1 \vee l_2 \vee l_3)$ has 3 distinct variables. Hence $E(Z_i) =$ .

# Approximate MaxSAT

Optimization Problem Given a rectified $3CNF$ formula $\phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, the aim is to find an assignment which satisfies the maximum number of clauses. Value of solution is number of clauses satisfied. We approximate MaxSAT.

- An assignment is uniformly randomly generated.
- Random variable $Z$ gives the number of satisfied clauses.
- Indicator random variable $Z_i = 1$ iff clause $C_i$ is satisfied. Hence $Z = \Sigma_{i=1}^{k} Z_i$.
- In rectified $3CNF$, each clause $C_i = (l_1 \vee l_2 \vee l_3)$ has 3 distinct variables. Hence $E(Z_i) = 7/8$.

# Approximate MaxSAT

Optimization Problem Given a rectified $3CNF$ formula $\phi$ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, the aim is to find an assignment which satisfies the maximum number of clauses. Value of solution is number of clauses satisfied. We approximate MaxSAT.

- An assignment is uniformly randomly generated.
- Random variable $Z$ gives the number of satisfied clauses.
- Indicator random variable $Z_i = 1$ iff clause $C_i$ is satisfied. Hence $Z = \Sigma_{i=1}^{k} Z_i$.
- In rectified $3CNF$, each clause $C_i = (l_1 \vee l_2 \vee l_3)$ has 3 distinct variables. Hence $E(Z_i) = 7/8$.
- Calculate $E(Z) = E(\Sigma_{i=1}^{k} Z_i) = \Sigma_{i=1}^{k}\ E(Z_i) = 7k/8$.

# Multishot Algorithm

- One shot Algorithm: Expected to satisfy $7k/8$ clauses. (What does this mean?)

# Multishot Algorithm

- One shot Algorithm: Expected to satisfy $7k/8$ clauses. (What does this mean?)
- Multi shot Algorithm: Repeatedly generate assignment randomly and check how many clauses it satisfies. Stop when assigment satisfies at least $7k/8$ clauses.

# Multishot Algorithm

- One shot Algorithm: Expected to satisfy $7k/8$ clauses. (What does this mean?)
- Multi shot Algorithm: Repeatedly generate assignment randomly and check how many clauses it satisfies. Stop when assigment satisfies at least $7k/8$ clauses.
- Guaranteed to generate assignment which satisfies at least $7k/8$ clauses.

# Multishot Algorithm

- One shot Algorithm: Expected to satisfy $7k/8$ clauses. (What does this mean?)
- Multi shot Algorithm: Repeatedly generate assignment randomly and check how many clauses it satisfies. Stop when assigment satisfies at least $7k/8$ clauses.
- Guaranteed to generate assignment which satisfies at least $7k/8$ clauses.

# Multishot Algorithm

- One shot Algorithm: Expected to satisfy $7k/8$ clauses. (What does this mean?)
- Multi shot Algorithm: Repeatedly generate assignment randomly and check how many clauses it satisfies. Stop when assigment satisfies at least $7k/8$ clauses.
- Guaranteed to generate assignment which satisfies at least $7k/8$ clauses.
- What is the expected running time?

# Waiting Time Bound: Multiple trials till success

# Waiting Time Bound: Multiple trials till success

- Bernoulli trial A Random trial with outcomes $T, F$ with probability of success $p$. We consider a sequence of independant Bernoulli trials.

# Waiting Time Bound: Multiple trials till success

- Bernoulli trial A Random trial with outcomes $T, F$ with probability of success $p$. We consider a sequence of independant Bernoulli trials.
- What is the expected number of independant trials till we get a success?

# Waiting Time Bound: Multiple trials till success

- Bernoulli trial A Random trial with outcomes $T, F$ with probability of success $p$. We consider a sequence of independant Bernoulli trials.
- What is the expected number of independant trials till we get a success?
- Let $R$ be random variable giving number of trials till success. $E[R] = 1 * p + 2 * (1-p) * p + 3 * (1-p)^2 * p + \ldots$
- This simplifies to $E[R] = 1/p$.

# MaxSAT: Expected Running Time

# MaxSAT: Expected Running Time

- Let $p_j$ be the probability that a random assignment satisfies exactly $j$ clauses.
  Let $p$ be the probability that a random assignment satisfies at least $7k/8$ clauses. Hence $p = \Sigma_{j \geq 7k/8}\ p_j$.

# MaxSAT: Expected Running Time

- Let $p_j$ be the probability that a random assignment satisfies exactly $j$ clauses.
  Let $p$ be the probability that a random assignment satisfies at least $7k/8$ clauses. Hence $p = \Sigma_{j \geq 7k/8}\ p_j$.
- Then $E(Z) =$

# MaxSAT: Expected Running Time

- Let $p_j$ be the probability that a random assignment satisfies exactly $j$ clauses.
  Let $p$ be the probability that a random assignment satisfies at least $7k/8$ clauses. Hence $p = \Sigma_{j \geq 7k/8}\ p_j$.
- Then $E(Z) =$

$$\frac{7}{8}k = \sum_{j=0}^{k} jp_j = \sum_{j<7k/8} jp_j + \sum_{j \geq 7k/8} jp_j.$$

# MaxSAT: Expected Running Time

- Let $p_j$ be the probability that a random assignment satisfies exactly $j$ clauses.
  Let $p$ be the probability that a random assignment satisfies at least $7k/8$ clauses. Hence $p = \Sigma_{j \geq 7k/8}\ p_j$.
- Then $E(Z) =$

$$\frac{7}{8}k = \sum_{j=0}^{k} jp_j = \sum_{j<7k/8} jp_j + \sum_{j \geq 7k/8} jp_j.$$

- Simplifying (see KT 13.5 (page 727)), we get $p \geq 1/(8k)$.

# MaxSAT: Expected Running Time

- Let $p_j$ be the probability that a random assignment satisfies exactly $j$ clauses.
  Let $p$ be the probability that a random assignment satisfies at least $7k/8$ clauses. Hence $p = \Sigma_{j \geq 7k/8}\ p_j$.
- Then $E(Z) =$

$$\frac{7}{8}k = \sum_{j=0}^{k} jp_j = \sum_{j<7k/8} jp_j + \sum_{j \geq 7k/8} jp_j.$$

- Simplifying (see KT 13.5 (page 727)), we get $p \geq 1/(8k)$.
- Hence, using waiting time bound, the expected number of trials to get the required assignment is $\leq (8k)$.

# MaxSAT: Expected Running Time

- Let $p_j$ be the probability that a random assignment satisfies exactly $j$ clauses.
  Let $p$ be the probability that a random assignment satisfies at least $7k/8$ clauses. Hence $p = \Sigma_{j \geq 7k/8}\ p_j$.
- Then $E(Z) =$

$$\frac{7}{8}k = \sum_{j=0}^{k} jp_j = \sum_{j<7k/8} jp_j + \sum_{j \geq 7k/8} jp_j.$$

- Simplifying (see KT 13.5 (page 727)), we get $p \geq 1/(8k)$.
- Hence, using waiting time bound, the expected number of trials to get the required assignment is $\leq (8k)$.

**(13.16)** *There is a randomized algorithm with polynomial expected running time that is guaranteed to produce a truth assignment satisfying at least a $\frac{7}{8}$ fraction of all clauses.*

# Derandomization using conditional expectations

Consider a formula $\phi$ with $k$ clauses where each clause has at most 3 literals (non-overlapping).

- $E(\phi) = \Sigma_{C \in \phi} (1 - 1/(2^{|C|})$.

# Derandomization using conditional expectations

Consider a formula $\phi$ with $k$ clauses where each clause has at most 3 literals (non-overlapping).

- $E(\phi) = \Sigma_{C \in \phi} (1 - 1/(2^{|C|})$.
- For a variable $x$, let $\phi_x = \phi[1/x]$ be simplified. Similarly $\phi_{\neg x}$.

# Derandomization using conditional expectations

Consider a formula $\phi$ with $k$ clauses where each clause has at most 3 literals (non-overlapping).

- $E(\phi) = \Sigma_{C \in \phi} (1 - 1/(2^{|C|})$.
- For a variable $x$, let $\phi_x = \phi[1/x]$ be simplified. Similarly $\phi_{\neg x}$.
- Let $\#_x$ be number of "true" clauses in $\phi_x$. Similarly, $\#_{\neg x}$

# Derandomization using conditional expectations

Consider a formula $\phi$ with $k$ clauses where each clause has at most 3 literals (non-overlapping).

- $E(\phi) = \Sigma_{C \in \phi} (1 - 1/(2^{|C|})$.
- For a variable $x$, let $\phi_x = \phi[1/x]$ be simplified. Similarly $\phi_{\neg x}$.
- Let $\#_x$ be number of "true" clauses in $\phi_x$. Similarly, $\#_{\neg x}$
- $E(\phi) = 1/2 \cdot (E(\phi_x) + \#_x)) + 1/2 \cdot (E(\phi_{\neg x}) + \#_{\neg x}))$

# Derandomization using conditional expectations

Consider a formula $\phi$ with $k$ clauses where each clause has at most 3 literals (non-overlapping).

- $E(\phi) = \Sigma_{C \in \phi} (1 - 1/(2^{|C|})$.
- For a variable $x$, let $\phi_x = \phi[1/x]$ be simplified. Similarly $\phi_{\neg x}$.
- Let $\#_x$ be number of "true" clauses in $\phi_x$. Similarly, $\#_{\neg x}$
- $E(\phi) = 1/2 \cdot (E(\phi_x) + \#_x)) + 1/2 \cdot (E(\phi_{\neg x}) + \#_{\neg x}))$

## Algorithm

1. Pick variable $x$.
2. Compute $E_1 = E(\phi)$, $E_2 = (E(\phi_x) + \#_x))$ and $E_3 = (E(\phi_{\neg x}) + \#_{\neg x}))$.
3. If $E_1 < E_2$ set $x = 1$ otherwise $x = 0$.
4. Goto (1) till variables are unassigned.

# Example

- $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
- $\phi_{x_1} = (1 \vee \neg x_2 \vee \neg x_3) \wedge (0 \vee x_2 \vee x_3) \wedge (1 \vee x_2 \vee x_3)$
  $= (x_2 \vee x_3)$
- $\#_{x_1} = 2.$
- $E(\phi_{x_1}) = (1\ (1/(2^2))) = 3/4.$

$\phi_{\neg x} = (\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3)$

$\#_{\neg x} = 1$

# Randomized Quicksort

- Guaranteed to produce sorted array for any input

# Randomized Quicksort

- Guaranteed to produce sorted array for any input
- Expected execution time $O(n \cdot lg(n))$ for any input.

# Randomized Quicksort

- Guaranteed to produce sorted array for any input
- Expected execution time $O(n \cdot lg(n))$ for any input.
- Same as quicksort with modified selection of pivot.

# Randomized Quicksort

- Guaranteed to produce sorted array for any input
- Expected execution time $O(n \cdot lg(n))$ for any input.
- Same as quicksort with modified selection of pivot.
- Central Pivot A pivot which split $S$ in $S^+$ and $S^-$ such that $|S^+| \geq 1/4 \cdot |S|$ and $|S^-| \geq 1/4 \cdot |S|$.

# Randomized Quicksort

- Guaranteed to produce sorted array for any input
- Expected execution time $O(n \cdot lg(n))$ for any input.
- Same as quicksort with modified selection of pivot.
- Central Pivot A pivot which split $S$ in $S^+$ and $S^-$ such that $|S^+| \geq 1/4 \cdot |S|$ and $|S^-| \geq 1/4 \cdot |S|$.
- Repeats random selection of pivot till a central pivot is found. Then partitions the array.

# Algorithm

```
Modified Quicksort(S):
If |S| ≤ 3 then
    Sort S
    Output the sorted list
Endif
Else
    While no central splitter has been found
        Choose a splitter a_i ∈ S uniformly at random
        For each element a_j of S
            Put a_j in S^- if a_j < a_i
            Put a_j in S^+ if a_j > a_i
        Endfor
        If |S^-| ≥ |S|/4 and |S^+| ≥ |S|/4 then
            a_i is a central splitter
        Endif
    Endwhile
    Recursively call Quicksort(S^-) and Quicksort(S^+)
    Output the sorted set S^-, then a_i, then the sorted set S^+
Endif
```

# Analysis

**Theorem**

*Randomized partition step with central splitter takes Expected time $O(n)$ on any input.*

**Theorem**

*Randomized partition step with central splitter takes Expected time $O(n)$ on any input.*

- Hence, recurrence of Expected worst-case execution time of randomized quicksort is roughly
  $T(n) = T(n/(4/3)) + T(n/(1/4)) + \Theta(n)$.

**Theorem**

*Randomized partition step with central splitter takes Expected time $O(n)$ on any input.*

- Hence, recurrence of Expected worst-case execution time of randomized quicksort is roughly $T(n) = T(n/(4/3)) + T(n/(1/4)) + \Theta(n)$.
- Solution using recursion tree is $T(n) = O(n \cdot lg(n))$.

# Analysis (2)

**Theorem**

*Randomized partition step with central splitter takes Expected time $O(n)$ on any input.*

# Analysis (2)

**Theorem**

*Randomized partition step with central splitter takes Expected time $O(n)$ on any input.*

- Given array $S$, Half of its elements when chosen as pivot will give central pivot.

# Analysis (2)

**Theorem**

*Randomized partition step with central splitter takes Expected time $O(n)$ on any input.*

- Given array $S$, Half of its elements when chosen as pivot will give central pivot.
- Hence for a random choice of pivot $a_i \in S$, the probability that $a_i$ is a central pivot is $p = 1/2$.

# Analysis (2)

**Theorem**

*Randomized partition step with central splitter takes Expected time $O(n)$ on any input.*

- Given array $S$, Half of its elements when chosen as pivot will give central pivot.
- Hence for a random choice of pivot $a_i \in S$, the probability that $a_i$ is a central pivot is $p = 1/2$.
- Hence, the waiting time bound states that the expected number of iterations to find the central pivot is $1/p$ i.e. 2.

# Randomized Complexity Classes

## Class $\mathbb{ZPP}$

$L \in \mathbb{ZPP}$ if there exists a randomized algorithm $M(x, y)$ which is polynomial time in expectation such that

- $x \in L \Rightarrow Pr_y[M(x, y) = 1] \geq \alpha$ with $\alpha = 1$.
- $x \notin L \Rightarrow Pr_y[M(x, y) = 1] \leq \beta$ with $\beta = 0$.
- Hence, the randomized algorithm neither produces false negative nor false positive.

# Randomized Complexity Classes

## Class $\mathbb{ZPP}$

$L \in \mathbb{ZPP}$ if there exists a randomized algorithm $M(x, y)$ which is polynomial time in expectation such that

- $x \in L \Rightarrow Pr_y[M(x, y) = 1] \geq \alpha$ with $\alpha = 1$.
- $x \notin L \Rightarrow Pr_y[M(x, y) = 1] \leq \beta$ with $\beta = 0$.
- Hence, the randomized algorithm neither produces false negative nor false positive.

## Other Randomized Complexity Classes

# Randomized Complexity Classes

## Class $\mathbb{ZPP}$

$L \in \mathbb{ZPP}$ if there exists a randomized algorithm $M(x, y)$ which is polynomial time in expectation such that

- $x \in L \Rightarrow Pr_y[M(x, y) = 1] \geq \alpha$ with $\alpha = 1$.
- $x \notin L \Rightarrow Pr_y[M(x, y) = 1] \leq \beta$ with $\beta = 0$.
- Hence, the randomized algorithm neither produces false negative nor false positive.

## Other Randomized Complexity Classes

- Choosing $\alpha = 2/3,\ \beta = 0$ we get the class $\mathbb{RP}$.

# Randomized Complexity Classes

## Class $\mathbb{ZPP}$

$L \in \mathbb{ZPP}$ if there exists a randomized algorithm $M(x, y)$ which is polynomial time in expectation such that

- $x \in L \Rightarrow Pr_y[M(x, y) = 1] \geq \alpha$ with $\alpha = 1$.
- $x \notin L \Rightarrow Pr_y[M(x, y) = 1] \leq \beta$ with $\beta = 0$.
- Hence, the randomized algorithm neither produces false negative nor false positive.

## Other Randomized Complexity Classes

- Choosing $\alpha = 2/3,\ \beta = 0$ we get the class $\mathbb{RP}$.
- Choosing $\alpha = 2/3,\ \beta = 1/3$ we get the class $\mathbb{BPP}$.

# Randomized Complexity Classes

## Class $\mathbb{ZPP}$

$L \in \mathbb{ZPP}$ if there exists a randomized algorithm $M(x, y)$ which is polynomial time in expectation such that

- $x \in L \Rightarrow Pr_y[M(x, y) = 1] \geq \alpha$ with $\alpha = 1$.
- $x \notin L \Rightarrow Pr_y[M(x, y) = 1] \leq \beta$ with $\beta = 0$.
- Hence, the randomized algorithm neither produces false negative nor false positive.

## Other Randomized Complexity Classes

- Choosing $\alpha = 2/3,\ \beta = 0$ we get the class $\mathbb{RP}$.
- Choosing $\alpha = 2/3,\ \beta = 1/3$ we get the class $\mathbb{BPP}$.
- Clearly, $\mathbb{P} \subseteq \mathbb{ZPP} \subseteq \mathbb{RP} \subseteq \mathbb{BPP}$.

# Randomized Complexity Classes

## Class $\mathbb{ZPP}$

$L \in \mathbb{ZPP}$ if there exists a randomized algorithm $M(x, y)$ which is polynomial time in expectation such that

- $x \in L \Rightarrow Pr_y[M(x, y) = 1] \geq \alpha$ with $\alpha = 1$.
- $x \notin L \Rightarrow Pr_y[M(x, y) = 1] \leq \beta$ with $\beta = 0$.
- Hence, the randomized algorithm neither produces false negative nor false positive.

## Other Randomized Complexity Classes

- Choosing $\alpha = 2/3,\ \beta = 0$ we get the class $\mathbb{RP}$.
- Choosing $\alpha = 2/3,\ \beta = 1/3$ we get the class $\mathbb{BPP}$.
- Clearly, $\mathbb{P} \subseteq \mathbb{ZPP} \subseteq \mathbb{RP} \subseteq \mathbb{BPP}$.
- Open Problem: $\mathbb{P} = \mathbb{BPP}$?