Randomized Algorithms

(PhD Course)

Fundamentals

Mohammad Farshi

Department of Computer Science,
Yazd University

1394-2

Elementary Probability Theory

- S: sample space
- Probability distribution: $Prob : P(S) \to [0, 1]$, s.t.
  - $Prob(\{x\}) \geq 0, \forall x \in S$,
  - $Prob(S) = 1$,
  - $Prob(A \cup B) = Prob(A) + Prob(B), \forall A, B \subset S$ with $A \cap B = \emptyset$.
- $A \subset S$: event
- $Prob(A)$: probability of event A.
- Uniform probability distribution on S:

$$Prob(\{x\}) = \frac{1}{|S|}, \forall x \in S.$$

.

Elementary Probability Theory

- $Prob(\emptyset) = 0.$.
- $Prob(S - A) = 1 - Prob(A), \forall A \subset S.$
- $Prob(A) \leq Prob(B), \forall A \subset B$
- $Prob(A \cup B) = Prob(A) + Prob(B) - Prob(A \cap B)$
- $Prob(A) = \sum_{x \in A} Prob(\{x\}), \forall A \subset S.$
- Probability of A given B:

$$Prob(A|B) = \frac{Prob(A \cap B)}{Prob(B)}.$$

- Two events $A, B \subset S$ are independent, if .

$$Prob(A \cap B) = Prob(A) \times Prob(B).$$

Elementary Probability Theory

- Probabilistic Experiment: like, Flipping a coin.
- A (discrete) random variable on a sample space S: a function X from S to $\mathbb{R}$
- Example: $S = \{head, tail\}$, $X(head) = 1, X(tail) = 0$.
- Example: Consider a ramdomized algorithm. S: different reuns of the algorithm on a same input. X: running time of each run.
- $Event(X = z) = \{s \in S | X(s) = z\}$.
- The probability density function of the random variable X: $f_X : \mathbb{R} \to [0, 1]$, $f_X(z) = Prob(Event(X = z))$.

Elementary Probability Theory

- The distribution function of X is a function $Dis_X : \mathbb{R} \to [0, 1]$, defined by

$$Dis_X(z) = Prob(X \leq z)$$

- The expectation of X (or the expected value of X) is

$$E[X] = \sum_x x \times Prob(X = x).$$

- $(S, Prob)$: a finite probability space.
  X: a random variable on S.

$$E[X] = \sum_{s \in S} X(s) \times Prob(\{s\}).$$

Elementary Probability Theory

- The expectation of X (or the expected value of X) is

$$E[X] = \sum_{x} x \times Prob(X = x).$$

- $(S, Prob)$: a finite probability space.
  X: a random variable on S.

$$E[X] = \sum_{s \in S} X(s) \times Prob(\{s\}).$$

- $E[a \times X + b] = a \times E[X] + b$ (weak linearity of expectation).
- $E[X + Y] = E[X] + E[Y]$ (linearity of expectation).

Models of Randomized Algorithms

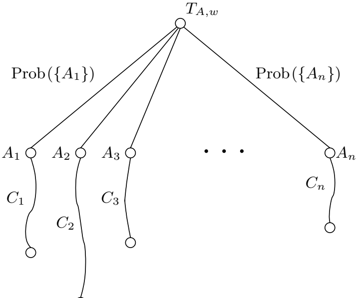
stochastic algorithms vs. randomized algorithms

- stochastic algorithm: use random choices in algorithm.
- stichastic algorithm can behaves “poorl” on some input instances when the algorithm is “good” for most inputs.
- The randomized algorithms: an stochastic algorithms that are not allowed to behave poorly on any input instance.
- one has to investigate the behavior of a randomized algorithm on any feasible input.
- Therefore, it does not make any sense to consider here probability distributions on input sets.
- The only source of randomness under consideration is the randomized control of the algorithm itself.

Models of Randomized Algorithms

1st Model

- $\{A_1, A_2, \ldots, A_n\}$: deterministic algorithms
- for any input w, A chooses an A_i at random and lets A_i work on w.
- C_i: the computation of A_i on w.
- A chooses A_i with probability $Prob(\{A_i\})$.
- $Z(C_i) = Time(C_i)$



Models of Randomized Algorithms

1st Model

- $\{A_1, A_2, \ldots, A_n\}$: deterministic algorithms
- for any input w, A chooses an A_i at random and lets A_i work on w.
- C_i: the computation of A_i on w.
- A chooses A_i with probability $Prob(\{A_i\})$.
- $Z(C_i) = Time(C_i)$

the expected time complexity of A on w:

$$\begin{aligned} Exp-Time_A(w) = E[Z] &= \sum_{i=1}^{n} Prob(\{C_i\}) \times Z(C_i) \\ &= \sum_{i=1}^{n} Prob(\{C_i\}) \times Time(C_i). \end{aligned}$$

1st Model

- $\{A_1, A_2, \ldots, A_n\}$: deterministic algorithms
- for any input w, A chooses an A_i at random and lets A_i work on w.
- C_i: the computation of A_i on w.
- A chooses A_i with probability $Prob(\{A_i\})$.
- $Z(C_i) = Time(C_i)$

expected time complexity

The expected time complexity of A
$Exp - Time_A(n) =$
$\max\{Exp - Time_A(w) |$ the length of w is $n\}$

1st Model– “reliability” of a randomized algorithm

- “reliability” of a randomized algorithm A on input w:

$$X(C_i) = \begin{cases} 1 & \text{if } C_i \text{ is correct on } w \\ 0 & \text{if } C_i \text{ is wrong on } w \end{cases}$$

$$\begin{aligned} E[X] &= \sum_{i=1}^{n} X(C_i) \times Prob(\{C_i\}) \\ &= \sum_{X(C_i)=1} 1 \times Prob(\{C_i\}) + \sum_{X(C_i)=0} 0 \times Prob(\{C_i\}) \\ &= Prob(Event(X = 1)) \\ &= \text{the probability that A computes the right result.} \end{aligned}$$

Error probability of A:

Models of Randomized Algorithms

1st Model– “reliability” of a randomized algorithm

- “reliability” of a randomized algorithm A on input w:

$$X(C_i) = \begin{cases} 1 & \text{if } C_i \text{ is correct on } w \\ 0 & \text{if } C_i \text{ is wrong on } w \end{cases}$$

Error probability of A:

$$Error_A(n) = \max\{Error_A(w) | \text{ the length of } w \text{ is } n\}$$

Models of Randomized Algorithms

Example: Randomized protocol

- No analysis on protocol complexity.
- For input (x, y), when $x = y$, the error prob. $= 0$.
- If $x \neq y$, $X(C_p) = \begin{cases} 1 & \text{if } p \text{ is good.} \\ 0 & \text{if } p \text{ is bad.} \end{cases}$
- $Prob(C_p) = \frac{1}{PRIM(n^2)}$.

Error probability of protocol:

$$Error_R(x, y) = 1 - E[X] \leq \frac{n-1}{Prim(n^2)} \leq \frac{2\log n}{n}$$

Error probability of modofied protocol (two primes):

$$Error_{R_2}(x, y) \leq \left(\frac{2\log n}{n}\right)^2$$

Example: Randomized protocol

- No analysis on protocol complexity.
- For input (x, y), when $x = y$, the error prob. $= 0$.
- If $x \neq y$, $X(C_p) = \begin{cases} 1 & \text{if } p \text{ is good.} \\ 0 & \text{if } p \text{ is bad.} \end{cases}$
- $Prob(C_p) = \frac{1}{PRIM(n^2)}$.

$$\begin{aligned} E[X] &= \sum_{p \in PRIM(n^2)} X(C_p) \times Prob(\{C_p\}) \\ &= \sum_{p \in PRIM(n^2)} X(C_p) \times \frac{1}{Prim(n^2)} \\ &= \frac{1}{Prim(n^2)} \times \sum_{p \text{ is good}} X(C_p) \\ &\geq \frac{1}{Prim(n^2)} \times \left(Prim(n^2) - (n-1)\right) \\ &= 1 - \frac{n-1}{Prim(n^2)}. \end{aligned}$$

Example: Randomized protocol

- No analysis on protocol complexity.
- For input (x, y), when $x = y$, the error prob. $= 0$.
- If $x \neq y$, $X(C_p) = \begin{cases} 1 & \text{if } p \text{ is good.} \\ 0 & \text{if } p \text{ is bad.} \end{cases}$
- $Prob(C_p) = \frac{1}{PRIM(n^2)}$.
  $E[X] = 1 - \frac{n-1}{Prim(n^2)}$.

Error probability of protocol:

$$Error_R(x, y) = 1 - E[X] \leq \frac{n-1}{Prim(n^2)} \leq \frac{2\log n}{n}$$

Error probability of modofied protocol (two primes):

$$Error_{R_2}(x, y) \leq \left(\frac{2\log n}{n}\right)^2$$

Example: Randomized protocol

- No analysis on protocol complexity.
- For input (x, y), when $x = y$, the error prob. $= 0$.
- If $x \neq y$, $X(C_p) = \begin{cases} 1 & \text{if } p \text{ is good.} \\ 0 & \text{if } p \text{ is bad.} \end{cases}$
- $Prob(C_p) = \frac{1}{PRIM(n^2)}$.
  $E[X] = 1 - \frac{n-1}{Prim(n^2)}$.

Error probability of protocol:

$$Error_R(x, y) = 1 - E[X] \leq \frac{n-1}{Prim(n^2)} \leq \frac{2\log n}{n}$$

Error probability of modofied protocol (two primes):

$$Error_{R_2}(x, y) \leq \left(\frac{2\log n}{n}\right)^2$$

Example: MAX-SAT

- Given a formula $\phi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$, in CNF over $\{x_1, \ldots, x_n\}$, find an assigment that maximized the satisfied clauses.
- RSAM Algorithm (Random Sampling): assign 0 and 1 with same probability to each variable.
- No error probability here.
- How good is the answer?

Example: MAX-SAT

- Given a formula $\phi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$, in CNF over $\{x_1, \ldots, x_n\}$, find an assigment that maximized the satisfied clauses.
- RSAM Algorithm (Random Sampling): assign 0 and 1 with same probability to each variable.
- No error probability here.
- How good is the answer?

$$Z_i(\alpha) = \begin{cases} 1 & F_i \text{ is satisfied by } \alpha \\ 0 & F_i \text{ is not satisfied by } \alpha \end{cases}$$

$Z = \sum_{i=1}^{m} Z_i$.

$E[Z] = E\left[\sum_{i=1}^{m} Z_i\right] = \sum_{i=1}^{m} E[Z_i]$.

$F_i = \ell_{i1} \vee \ell_{i2} \vee \ldots \vee \ell_{ik}$.

$E[Z_i] = 1 - \frac{1}{2^k} \geq \frac{1}{2}$.

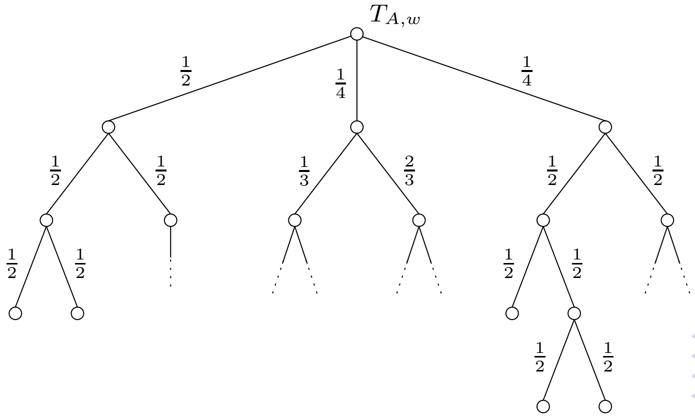
$E[Z] \geq \frac{m}{2}$.

The Second model

- Nondeterministic algorithm with probability distribution for any nondeterministic choice.
- Computation tree

The Second model

- Nondeterministic algorithm with probability distribution for any nondeterministic choice.
- Computation tree



The Second model

- Nondeterministic algorithm with probability distribution for any nondeterministic choice.
- Computation tree
- $S_{A,w}$ = all runs of A on w.
- If $C \in S_{A,w}$ then $Prob(C)$ = product of label of all edges on the path.
- The second model is a generalization of the first model.
- The second model is used for algorithms that repeatedly makes a random choice.

The Second model

- Nondeterministic algorithm with probability distribution for any nondeterministic choice.
- Computation tree
- $S_{A,w}$ = all runs of A on w.
- If $C \in S_{A,w}$ then $Prob(C)$ = product of label of all edges on the path.
- The second model is a generalization of the first model.
- The second model is used for algorithms that repeatedly makes a random choice.

The Second model

- Nondeterministic algorithm with probability distribution for any nondeterministic choice.
- Computation tree
- $S_{A,w} =$ all runs of A on w.
- If $C \in S_{A,w}$ then $Prob(C) =$ product of label of all edges on the path.
- The second model is a generalization of the first model.
- The second model is used for algorithms that repeatedly makes a random choice.

The Second model

- Nondeterministic algorithm with probability distribution for any nondeterministic choice.
- Computation tree
- $S_{A,w}$ = all runs of A on w.
- If $C \in S_{A,w}$ then $Prob(C)$ = product of label of all edges on the path.
- The second model is a generalization of the first model.
- The second model is used for algorithms that repeatedly makes a random choice.

The Second model

- Nondeterministic algorithm with probability distribution for any nondeterministic choice.
- Computation tree
- $S_{A,w} =$ all runs of A on w.
- If $C \in S_{A,w}$ then $Prob(C) =$ product of label of all edges on the path.
- The second model is a generalization of the first model.
- The second model is used for algorithms that repeatedly makes a random choice.

Example: Randomized Quick Sort

- Input: A set A of elements.
- Step 1: If A contains one elements return it.
- Step 2: If $|A| \geq 2$, choose a random elements $a \in A$
- Split A to two sets, elements bigger than a and elemets less than a.
- Recursively sort each subset and the result is the first subset, a, the second subset.

Analysis

- The error probability is 0.
- Step 2: $|A| - 1$ comparisons
- If the algorithm chooses first or last element, there is $n-1$ recursive calls, $O(n^2)$ comparisons.
- If the algorithm chooses median at each step, the time complexity is $O(n \log n)$.

Example: Randomized Quick Sort

- Input: A set A of elements.
- Step 1: If A contains one elements return it.
- Step 2: If $|A| \geq 2$, choose a random elements $a \in A$
- Split A to two sets, elements bigger than a and elemets less than a.
- Recursively sort each subset and the result is the first subset, a, the second subset.

Analysis

- Except median, there are several other good choices: if algorithm chooses αnth element, for some constant $\alpha < 1$, the recursion changes to $T(n) \leq T(\alpha n) + T((1-\alpha)n) + n - 1$ which solves to $T(n) \in O(n \log n)$.

Example: Randomized Quick Sort-Analysis

- The computation tree of RQS is large and irregular.
- C: a run of RQS on A, $X(C) = \#$ comparisons in C.
- Let $s_1, \ldots, s_n$ is output of RQS.
- $X_{ij}(C) = \begin{cases} 1 & s_i, s_j \text{ compared in } C \\ 0 & \text{otherwise.} \end{cases}$
- $T(C) = \sum_{i=1}^{n} \sum_{j>i} X_{ij}(C) = \#$ comparisons in C.
- $E[T] = Exp-Time_{RQS}(A)$.
- $E[T] = \sum_{i=1}^{n} \sum_{j>i} E[X_{ij}]$.
- $p_{ij} =$ probability that s_i and s_j are compared.
- $E[X_{ij}] = p_{ij}$.

Example: Randomized Quick Sort-Analysis

- p_{ij} = probability that s_i and s_j are compared.
- $E[X_{ij}] = p_{ij}$.
- p_{ij} = probability that s_i of s_j is choosen as pivot before any of Middle points.
- If an element from Left or right is choosen, it does not influence the comparison of s_i and s_j.
- $$p_{ij} = \frac{|\{s_i, s_j\}|}{|Middle \cup \{s_i, s_j\}|} = \frac{2}{j-i+1}.$$

Example: Randomized Quick Sort-Analysis

- $p_{ij} =$ probability that s_i and s_j are compared.
- $E[X_{ij}] = p_{ij}$.
- $p_{ij} =$ probability that s_i of s_j is choosen as pivot before any of Middle points.
- If an element from Left or right is choosen, it does not influence the comparison of s_i and s_j.
- $$p_{ij} = \frac{|\{s_i, s_j\}|}{|Middle \cup \{s_i, s_j\}|} = \frac{2}{j-i+1}.$$

$$\underbrace{s_1 \ldots s_{i-1}}_{\text{Left}} \; s_i \; \underbrace{s_{i+1}\, s_{i+2} \; \ldots \; s_{j-1}}_{\text{Middle}} \; s_j \; \underbrace{s_{j+1} \ldots s_n}_{\text{Right}}$$

Example: Randomized Quick Sort-Analysis

- $p_{ij} =$ probability that s_i and s_j are compared.
- $E[X_{ij}] = p_{ij}$.
- $p_{ij} =$ probability that s_i of s_j is choosen as pivot before any of Middle points.
- If an element from Left or right is choosen, it does not influence the comparison of s_i and s_j.
- $$p_{ij} = \frac{|\{s_i, s_j\}|}{|Middle \cup \{s_i, s_j\}|} = \frac{2}{j-i+1}.$$

$$\underbrace{\boxed{s_1 \ldots s_{i-1}}}_{\text{Left}}\boxed{s_i}\underbrace{\boxed{s_{i+1}\ s_{i+2} \quad \ldots \quad s_{j-1}}}_{\text{Middle}}\boxed{s_j}\underbrace{\boxed{s_{j+1} \ldots s_n}}_{\text{Right}}$$

Example: Randomized Quick Sort-Analysis

- $p_{ij} =$ probability that s_i and s_j are compared.
- $E[X_{ij}] = p_{ij}$.
- $p_{ij} =$ probability that s_i of s_j is choosen as pivot before any of Middle points.
- If an element from Left or right is choosen, it does not influence the comparison of s_i and s_j.
- $$p_{ij} = \frac{|\{s_i, s_j\}|}{|Middle \cup \{s_i, s_j\}|} = \frac{2}{j - i + 1}.$$

$$\underbrace{s_1 \ldots s_{i-1}}_{\text{Left}} \;\Big|\; s_i \;\Big|\; \underbrace{s_{i+1}\, s_{i+2} \quad \ldots \quad s_{j-1}}_{\text{Middle}} \;\Big|\; s_j \;\Big|\; \underbrace{s_{j+1} \ldots s_n}_{\text{Right}}$$

Example: Randomized Quick Sort-Analysis

- $E[T] = \sum_{i=1}^{n} \sum_{j>i} E[X_{ij}]$.
- p_{ij} = probability that s_i and s_j are compared.
- $E[X_{ij}] = p_{ij} = \frac{2}{j-i+1}$.

$$
\begin{aligned}
E[T] &= \sum_{i=1}^{n} \sum_{j>i} p_{ij} \\
&= \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \\
&\leq \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \\
&= 2nHar(n) = 2n \log n + \Theta(n).
\end{aligned}
$$

Classification of Randomized Algorithms

LAS VEGAS Algorithms

- Guarantee that output is correct (always).
- $Prob(A(x) = F(x)) = 1$
- We investigate expected time complexity of the algorithms.
- Different run lengths on different inputs (otherwise we can construct a deterministic algorithm).
- Examples: Randomized Quick sort, Randomized RSEL.
- There a variation that can output "I do not know".

LAS VEGAS Algorithms (2nd def.)

For some $0 < \varepsilon < 1$

- $Prob(A(x) = F(x)) \geq \varepsilon$
- $Prob(A(x) = "?") \leq 1 - \varepsilon$.

Classification of Randomized Algorithms

LAS VEGAS Algorithms

- Guarantee that output is correct (always).
- $Prob(A(x) = F(x)) = 1$
- We investigate expected time complexity of the algorithms.
- Different run lengths on different inputs (otherwise we can construct a deterministic algorithm).
- Examples: Randomized Quick sort, Randomized RSEL.
- There a variation that can output "I do not know".

LAS VEGAS Algorithms (2nd def.)

For some $0 < \varepsilon < 1$

- $Prob(A(x) = F(x)) \geq \varepsilon$
- $Prob(A(x) = "?") \leq 1 - \varepsilon$.

Example: LV_{10}

- We have two computers R_I and R_{II}
- $\{x_1, \ldots, x_{10}\}$ in R_I and $\{y_1, \ldots, y_{10}\}$ on R_{II}
- Question: Is there an index j s. t. $x_j = y_j$?
- Complexity of protocol: # exchanged bits.
- Any deterministic protocol needs $10n$ bits exchange.
- Here we show a Las Vegas protocol with complexity $n + \mathcal{O}(\log n)$.
- Note that previous protocol is not Las Vegas.

Example: LV$_{10}$: The ptorocol

- $p_1, \ldots, p_{10}$: 10 primes chosen randomly in PRIM(n^2).
- R_I computes $s_i = x_i$ mod $p_i, i = 1, \ldots, 10$.
- R_I sends $p_1, \ldots, p_{10}$ and $s_1, \ldots, s_{10}$ to R_{II}.
- R_{II} computes $q_i = y_i$ mod $p_i, i = 1, \ldots, 10$.
- If $s_i \neq q_i, \forall i = 1, \ldots, 10$, output NO.
- Else let j be the smallest index s. t. $s_j = q_j$, and R_{II} sends y_j and j to R_I.
- R_I compare x_j and y_j, if $x_j = y_j$ outputs Yes, otherwise outputs "?".

Protocol Complexity:

Communications: $p_1, \ldots, p_{10}, s_1, \ldots, s_10, j, y_j$= $n + \mathcal{O}(\log n)$ bits.

Example: LV_{10}: The ptorocol

- $p_1, \ldots, p_{10}$: 10 primes chosen randomly in PRIM(n^2).
- R_I computes $s_i = x_i$ mod $p_i, i = 1, \ldots, 10$.
- R_I sends $p_1, \ldots, p_{10}$ and $s_1, \ldots, s_{10}$ to R_{II}.
- R_{II} computes $q_i = y_i$ mod $p_i, i = 1, \ldots, 10$.
- If $s_i \neq q_i, \forall i = 1, \ldots, 10$, output NO.
- Else let j be the smallest index s. t. $s_j = q_j$, and R_{II} sends y_j and j to R_I.
- R_I compare x_j and y_j, if $x_j = y_j$ outputs Yes, otherwise outputs "?".

Protocol Complexity:

Communications: $p_1, \ldots, p_{10}, s_1, \ldots, s_10, j, y_j$= $n + \mathcal{O}(\log n)$ bits.

Example: LV_{10}: The ptorocol is Las Vegas

- Obviously the output of the algorithm is always correct.
- We shows that the probability condition is also holds.
- Case (i): $x_i \neq y_i, \forall i$:
  - The probability that $x_i \neq y_i \geq 1 - \frac{2\log n}{n}$.
  - Since the primes are chosen independently:

$$Prob((s_1, \ldots, s_{10}) \neq (q_1, \ldots, q_{10})) \geq \left(1 - \frac{2\log n}{n}\right)^{10}$$

  - So,

$$\begin{aligned} & Prob(LV_{10}((x_1, \ldots, x_{10}), (y_1, \ldots, y_{10})) = No) \geq \\ \geq \quad & \left(1 - \frac{2\log n}{n}\right)^{10} \\ \geq \quad & \left(1 - \frac{20\log n}{n}\right) \geq \frac{1}{2} \end{aligned}$$

Example: LV_{10}: The ptorocol is Las Vegas

- Case (ii): $x_i = y_i$, for some i:
  - Let j be the smallest index s.t. $x_j = y_j$.
  - If the protocol exchange some $i < j$ then the answer will be "?".
  - So, $\forall i \in \{1, \dots, j-1\}$ we must have $x_i \bmod p_i \neq y_i \bmod p_i$ (Event E_j).
  - If $j = 1$, the answer is always correct.
  - If $j > 1$, $$Prob(E_j) \geq$$ $$\geq \left(1 - \frac{2\log n}{n}\right)^{j-1}$$ $$\geq \left(1 - \frac{2(j-1)\log n}{n}\right)$$
  - The function takes its minimum at $j = 10$, so

$$Prob(LV_{10}((x_1, \dots, x_{10}), (y_1, \dots, y_{10})) = Yes) \geq 1 - \frac{18\log n}{n}.$$

LAS VEGAS* Algorithms

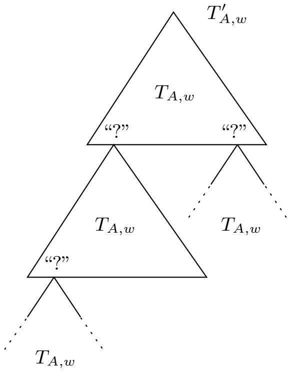
- $Prob(A(x) = F(x)) \to 1$ as $|x| \to \infty$

Are two definitions equivalent? (with and without "?")

- A= a Las Vegas algorithm with output "?".
- A': Runs A as far as its output is "?".
- The output of the algorithm is always correct.

Are two definitions equivalent? (with and without "?")

- A= a Las Vegas algorithm with output "?".
- A': Runs A as far as its output is "?".
- The output of the algorithm is always correct.



Are two definitions equivalent? (with and without "?")

- A= a Las Vegas algorithm with output "?".
- A': Runs A as far as its output is "?".
- The output of the algorithm is always correct.
- A' can have infinite computations.
- $Time_A(w)$= the worst-case complexity of A.
- The probability that A' stops in time $Time_A(w)$ is $\geq 1/2$.
- The probability that A' stops in time $2 \times Time_A(w)$ is $\geq 3/4$.
- The probability that A' stops in time $k \times Time_A(w)$ is $\geq 1 - 1/2^k$.
- Claim: $Exp - Time_{A'}(n) \in \mathcal{O}(Time_A(n))$.

Are two definitions equivalent? (with and without "?")

- Claim: $Exp - Time_{A'}(n) \in \mathcal{O}(Time_A(n))$.
  - Set_i= all computations that end exactly at ith run.
  - $S_{A',w} = \cup_1^{\infty} Set_i$, and $Set_i \cap Set_j = \emptyset$.
  - $\sum_{C \text{ ends before } k} Prob(\{C\}) \geq 1 - \frac{1}{2^k}$
  - So, $\sum_{C \in Set_i} Prob(\{C\}) \leq \frac{1}{2^{i-1}}$.

$$Exp - Time_{A'}(n) =$$

$$= \sum_{C \in S_{A',w}} Time_{A'}(C) \times Prob(\{C\})$$

$$= \sum_{i=1}^{\infty} \sum_{C \in Set_i} Time_{A'}(C) \times Prob(\{C\})$$

$$= \sum_{i=1}^{\infty} \sum_{C \in Set_i} i \times Time_A(w) \times Prob(\{C\})$$

$$= \sum_{i=1}^{\infty} i \times Time_A(w) \times \sum_{C \in Set_i} Prob(\{C\})$$

Are two definitions equivalent? (with and without "?")

- Claim: $Exp-Time_{A'}(n) \in \mathcal{O}(Time_A(n))$.
  - Set_i= all computations that end exactly at ith run.
  - $S_{A',w} = \cup_1^{\infty} Set_i$, and $Set_i \cap Set_j = \emptyset$.
  - $\sum_{C \text{ ends before } k} Prob(\{C\}) \geq 1 - \frac{1}{2^k}$
  - So, $\sum_{C \in Set_i} Prob(\{C\}) \leq \frac{1}{2^{i-1}}$.

$$
\begin{aligned}
& Exp-Time_{A'}(n) = \\
= \quad & \sum_{i=1}^{\infty} i \times Time_A(w) \times \sum_{C \in Set_i} Prob(\{C\}) \\
\leq \quad & \sum_{i=1}^{\infty} i \times Time_A(w) \times \frac{1}{2^{i-1}} \\
< \quad & 6 \times Time_A(w).
\end{aligned}
$$

Are two definitions equivalent? (with and without "?")

- Obviously any algorithm without "?" can be considered as an algorithm with "?".
- Reasons to convert a Las Vegas algorithm without "?" into a Las Vegas algorithm with output "?".
  - Consider algorithm A that mostly runs fast but sometime runs slow.
  - When a run take more than the time for fast run, stop it with output "?".
  - Claim: $2 \times Exp - Time_A(w)$ is a suitable upper bound for stopping.

Are two definitions equivalent? (with and without "?")

- To show that the new algorithm is Las Vegas, we have to show that $Prob(B(w) = "?") \leq 1/2$.
- Proof by contradiction:

$$
\begin{aligned}
Exp - Time_A(w) &= \sum_{C \in S_{A,w}} Time_A(C) \times Prob(\{C\}) \\
&= \sum_{C \in S_{A,w("?")}} Time_A(C) \times Prob(\{C\}) \\
&+ \sum_{C \in S_{A,w(F(w))}} Time_A(C) \times Prob(\{C\}) \\
&> \sum_{C \in S_{A,w("?")}} Time_A(C) \times Prob(\{C\}) \\
&> \sum_{C \in S_{A,w("?")}} (2 \times Exp - Time_A(w) + 1) \times Prob(\{C\}) \\
&> (2 \times Exp - Time_A(w) + 1) \times 1/2 \\
&= Exp - Time_A(w) + 1/2 \ \text{(Contradiction!)}
\end{aligned}
$$

One-Sided-Error Monte Carlo Alg.

- For decision problems.
- Let (σ, L) be a decision problem.
- Output "0" is almost correct, output "1" had bounded error.
- **Definition:** A is an one-sided-error Monte Carlo algorithm for L, 1MC algorithm for short, if
  - (i) $\forall x \in L$, $Prob(A(x) = 1) \geq 1/2$, and
  - (ii) $\forall x \notin L$, $Prob(A(x) = 0) = 1$.

1MC*

(i') $\forall x \in L, Prob(A(x) = 1)$ tends to 1 with growing $|x|$.

Similar to Las Vegas algorithms, one can decreases the error probability exponentially by repitations.

One-Sided-Error Monte Carlo Alg.

- After k repitition of A, the error probability decreases to 2^{-k}.
- Notation: A_k: k independents runs of A.

Bounded-Error Monte Carlo Algorithms

- **Definition:** A is a bounded-error Monte Carlo algorithm for F, 2MC algorithm for short, if $\exists\epsilon \in (0, 1/2]$ s.t. $\forall$ x of F, $Prob(A(x) = F(x)) \geq 1/2 + \epsilon$.
- If A is a Las Vegas or 1MC algorithm then A_2 is 2MC.

For any 2MC algorithm A and any positive integer t, let A_t denote the following randomized algorithm:

2MC Algorithm A_t

Input: x

- Step 1: Perform t independent runs of A on x and save the t computed results $\alpha_1, \ldots, \alpha_t$.
- Step 2: if there is an α that appears at least $\lceil t/2 \rceil$ times in the sequence $\alpha_1, \ldots, \alpha_t$, then
  output "α"

  else

  output "?"

Claim: The error probability of A_t tends to 0 with growing t.

Claim: The error probability of A_t tends to 0 with growing t.

- A_t computes a wrong result or "?" only if #correct results $< \lceil t/2 \rceil$.
- $pr_i(x)$= the probability that A_t computes the correct
- result in exactly i runs.

$$
\begin{aligned}
pr_i(x) &= \binom{t}{i} p^i (1-p)^{t-i} \\
&= \binom{t}{i} (p(1-p))^i (1-p)^{t-2i} \\
&= \binom{t}{i} \left(\left(\frac{1}{2}+\epsilon_x\right)\left(\frac{1}{2}-\epsilon_x\right)\right)^i \left(\frac{1}{2}-\epsilon_x\right)^{t-2i} \\
&= \binom{t}{i} \left(\frac{1}{4}+\epsilon_x^2\right)^i \left(\left(\frac{1}{2}-\epsilon_x\right)^2\right)^{t/2-i} \\
&\leq \binom{t}{i} \left(\frac{1}{4}-\epsilon^2\right)^{t/2}
\end{aligned}
$$

Claim: The error probability of A_t tends to 0 with growing t.

- A_t computes a wrong result or "?" only if #correct results $< \lceil t/2 \rceil$.
- $pr_i(x) \le \binom{t}{i} \left(\frac{1}{4} - \epsilon^2\right)^{t/2}$

$$
\begin{aligned}
Prob(A_t(x) = F(x)) &= 1 - \sum_{i=0}^{\lfloor t/2 \rfloor} pr_i(x) \\
&> 1 - \sum_{i=0}^{\lfloor t/2 \rfloor} \binom{t}{i} \left(\frac{1}{4} - \epsilon^2\right)^{t/2} \\
&= 1 - \left(\frac{1}{4} - \epsilon^2\right)^{t/2} \sum_{i=0}^{\lfloor t/2 \rfloor} \binom{t}{i} \\
&> 1 - \left(\frac{1}{4} - \epsilon^2\right)^{t/2} \times 2^t \\
&\ge 1 - (1 - 4\epsilon^2)^{t/2} \quad \text{(Done!)}
\end{aligned}
$$

Claim: The error probability of A_t tends to 0 with growing t.

- A_t computes a wrong result or "?" only if #correct results $< \lceil t/2 \rceil$.
- $Prob(A_t(x) = F(x)) \geq 1 - (1 - 4\epsilon^2)^{t/2}$
- Thus, if one looks for a k such that

  $$Prob(A_k(x) = F(x)) \geq 1 - \delta,$$

  for a chosen constant δ, it is sufficient to take

  $$k \geq \frac{2 \log \delta}{\log(1 - 4 \times \epsilon^2)}$$

- So, $Time_{A_k}(n) \in \mathcal{O}(Time_A(n))$.
- So, if A is asymptotically faster than any deterministic algorithm computing F, then A_k with an error probability below a chosen δ is also more efficient than any deterministic algorithm.

Unbounded-Error Monte Carlo Algorithms

- **Definition:** A randomized algorithm A is a (unbounded-error) Monte Carlo algorithm computing F, an MC algorithm, for short, if, for every input x of F, $Prob(A(x) = F(x)) > 1/2$.
- **Question:** What is the difference between 2MC and MC?
  - 2MC algorithms: the error probability to have a fixed distance from $1/2$ for any input.
  - MC algorithm: the distance between the error probability and $1/2$ may tends to 0 with growing input size $|x|$.
- **Question:** How many independent runs of A on x are necessary in order to get
$$Prob(A_k(x) = F(x)) > 1 - \delta$$
for a fixed chosen constant δ?

Unbounded-Error Monte Carlo Algorithms

- **Question:** How many independent runs of A on x are necessary in order to get
$$Prob(A_k(x) = F(x)) > 1 - \delta$$
for a fixed chosen constant δ?
- $Prob(A_t(x) = F(x)) \geq 1 - (1 - 4\epsilon_x^2)^{t/2}$
- Thus, if one looks for a k such that
$$Prob(A_k(x) = F(x)) \geq 1 - \delta,$$
for a chosen constant δ, it is sufficient to take
$$k = k(|x|) \geq \frac{2 \log \delta}{\log(1 - 4 \times \epsilon_x^2)} \geq (-2 \log \delta) \times 2^{2.|x|}.$$
- So, $Time_{A_k}(x) \geq (-2 \log \delta).2^{2.|x|}.Time_A(x)$.

MC Algorithms: Example

Protocol UMC

- Initial situation: R_I has n bits x, R_{II} has n bits y. The input (x, y) has to be accepted if and only if $x \neq y$.
- Phase 1: R_I uniformly chooses a number $j \in \{1, 2, \dots, n\}$ at random and sends j and the bit x_j to R_{II}.
- R_{II} compares x_j with y_j. If $x_j \neq y_j$, R_{II} accepts the input (x, y). If $x_j = y_j$, then R_{II} accepts (x, y) with probability $1/2 - 1/2n$, and rejects (x, y) with probability $1/2 + 1/2n$.

Protocol Complexity

$\lceil \log_2(n + 1) \rceil + 1$

MC Algorithms: Example

Protocol UMC

- Initial situation: R_I has n bits x, R_{II} has n bits y. The input (x, y) has to be accepted if and only if $x \neq y$.
- Phase 1: R_I uniformly chooses a number $j \in \{1, 2, \ldots, n\}$ at random and sends j and the bit x_j to R_{II}.
- R_{II} compares x_j with y_j. If $x_j \neq y_j$, R_{II} accepts the input (x, y). If $x_j = y_j$, then R_{II} accepts (x, y) with probability $1/2 - 1/2n$, and rejects (x, y) with probability $1/2 + 1/2n$.

Protocol Complexity

$\lceil \log_2(n + 1) \rceil + 1$

Claim: UMC Protocol is a Monte Carlo Protocol.

Case 1: $x = y$

C_{il} = the computation in which R_I chooses the number i at random in Phase 1 and R_{II} outputs l.

$Prob(C_{i0}) = \frac{1}{n} \times \left(\frac{1}{2} + \frac{1}{2n}\right)$

$Prob(C_{i1}) = \frac{1}{n} \times \left(\frac{1}{2} - \frac{1}{2n}\right)$

$A_0 = \{C_{i0} | 1 \leq i \leq n\}$: correct answer

$Prob(A_0) = \sum_{i=1}^{n} \frac{1}{n} \times \left(\frac{1}{2} + \frac{1}{2n}\right) = n \times \frac{1}{n} \times \left(\frac{1}{2} + \frac{1}{2n}\right) > \frac{1}{2}$.

Case 2: $x \neq y$

$\exists j \in \{1, 2, \ldots, n\}$ s.t. $x_j \neq y_j$

Worst case: Only one j exists with $x_j \neq y_j$.

$A_1 = \{C_j\} \cup \{C_{i1} | 1 \leq i \leq n, i \neq j\}$

$$Prob(A_1) = Prob(C_j) + \sum_{i=1, i\neq j}^{n} Prob(C_{i1})$$

$$= \frac{1}{n} + \sum_{i=1, i\neq j}^{n} \frac{1}{n} \times \left(\frac{1}{2} - \frac{1}{2n}\right) = \frac{1}{2} + \frac{1}{2n^2} > \frac{1}{2}.$$

Yazd Univ.

Randomized Algorithms-Fundamentals

Elementary Probability Theory

Models of Randomized Algorithms

Las Vegas Algorithms

Monte Carlo Algorithms

One-Sided-Error Monte Carlo Algorithms

Bounded-Error Monte Carlo Algorithms

Unbounded-Error Monte Carlo Algorithms

Claim: UMC Protocol is a Monte Carlo Protocol.

Case 1: $x = y$

C_{il} = the computation in which R_I chooses the number i at random in Phase 1 and R_{II} outputs l.

$Prob(C_{i0}) = \frac{1}{n} \times (\frac{1}{2} + \frac{1}{2n})$

$Prob(C_{i1}) = \frac{1}{n} \times (\frac{1}{2} - \frac{1}{2n})$

$A_0 = \{C_{i0} | 1 \leq i \leq n\}$: correct answer

$Prob(A_0) = \sum_{i=1}^{n} \frac{1}{n} \times (\frac{1}{2} + \frac{1}{2n}) = n \times \frac{1}{n} \times (\frac{1}{2} + \frac{1}{2n}) > \frac{1}{2}$.

Case 2: $x \neq y$

$\exists j \in \{1, 2, \ldots, n\}$ s.t. $x_j \neq y_j$

Worst case: Only one j exists with $x_j \neq y_j$.

$A_1 = \{C_j\} \cup \{C_{i1} | 1 \leq i \leq n, i \neq j\}$

$$Prob(A_1) = Prob(C_j) + \sum_{i=1, i\neq j}^{n} Prob(C_{i1})$$

$$= \frac{1}{n} + \sum_{i=1, i\neq j}^{n} \frac{1}{n} \times \left(\frac{1}{2} - \frac{1}{2n}\right) = \frac{1}{2} + \frac{1}{2n^2} > \frac{1}{2}.$$

Classification of Randomized Algorithms

for Optimization Problems

- We gave a classification of randomized algorithms for solving decision problems and for computing functions.
- For optimization problems, it is different. One can execute k runs of a randomized algorithm for an optimization problem and select the best one with respect to the optimization goal.

- A: computes an optimal solution only for an input x with probability $\geq \frac{1}{|x|}$.
- $A_{|x|}$: execute $|x|$ independent runs of A on x and then take the best output.
- Probability that $A_{|x|}$ **does not** find any optimal solution $\leq \left(1 - \frac{1}{|x|}\right)^{|x|} < \frac{1}{e}$.

Classification of Randomized Algorithms

for Optimization Problems

- We gave a classification of randomized algorithms for solving decision problems and for computing functions.
- For optimization problems, it is different. One can execute k runs of a randomized algorithm for an optimization problem and select the best one with respect to the optimization goal.

- A: computes an optimal solution only for an input x with probability $\geq \frac{1}{|x|}$.
- $A_{|x|}$: execute $|x|$ independent runs of A on x and then take the best output.
- Probability that $A_{|x|}$ **does not** find any optimal solution $\leq \left(1 - \frac{1}{|x|}\right)^{|x|} < \frac{1}{e}$.

Classification of Randomized Algorithms
for Optimization Problems

- We gave a classification of randomized algorithms for solving decision problems and for computing functions.
- For optimization problems, it is different. One can execute k runs of a randomized algorithm for an optimization problem and select the best one with respect to the optimization goal.

Time Complexity of $A_{|x|}$:

$|x| \times$ the time complexity of A.

For Optimization Problems

- the task is not necessarily to find an optimal solution, but one is usually satisfied with a feasible solution whose cost (quality) does not differ too much from the cost (quality) of an optimal solution.
- In such a case one can be looking for the probability of finding a relatively good situation, which leads to another classification of randomized algorithms.
- Solving an optimization problem cannot in general be considered as computing a function.
- Solving an optimization problem: computing a relation R in the sense that, for a given x, it is sufficient to find a y such that $(x, y) \in R$.
- There can exist many optimal solutions for an input x, and we are satisfied with any one of them.

Classification of Randomized Algorithms

for Optimization Problems

Optimization Problem:

- L: problem instances.
- $M(x)$: feasible solutions for input $x \in L$.
- cost: cost function
- goal: maximum/minimum
- A feasible solution $\alpha \in M(x)$ is **optimal** for x if $cost(\alpha, x) = goal\{cost(\beta, x) | \beta \in M(x)\}$.

Solving optimization problems

- **Naive Algorithm:** compute cost of all feasible solutions from $M(x)$, and pick the best one.
- **Problem:** The cardinality of $M(x)$ can be very large that it is practically impossible to generate $M(x)$.

Optimization Problem:

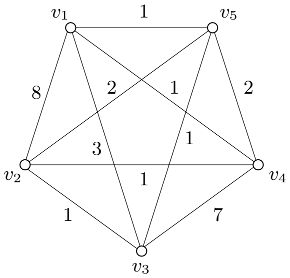
- L: problem instances.
- $M(x)$: feasible solutions for input $x \in L$.
- cost: cost function
- goal: maximum/minimum
- A feasible solution $\alpha \in M(x)$ is **optimal** for x if $cost(\alpha, x) = goal\{cost(\beta, x) | \beta \in M(x)\}$.

Solving optimization problems

- **Naive Algorithm:** compute cost of all feasible solutions from $M(x)$, and pick the best one.
- **Problem:** The cardinality of $M(x)$ can be very large that it is practically impossible to generate $M(x)$.

Example: Traveling Salesman Problem (TSP)

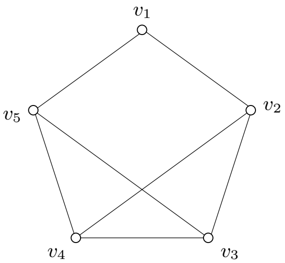
- Input: An edge weighted complete graph (G, c)
- $M(G, c)$: the set of all Hamiltonian cycles (i.e. $v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1}$ of vertices) of G.
- $cost(v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1}) = \sum_{j=1}^{n} c(\{v_{i_j}, v_{i_{(j \bmod n)+1}}\})$.
- goal: minimum



Classification of Randomized Algorithms
for Optimization Problems

Example: minimum vertex cover problem (MIN-VCP)

- Input: A graph $G(V, E)$
- $M(G)$: the set of all vertex covers of G.
  $U \subseteq V$ is a vertex cover of $G = (V, E)$, if every edge from E is incident to at least one vertex from U.
- $cost$ = # vertices in a vertex cover.
- goal: minimum if every edge from E is incident to at least one vertex from U.



Example: maximum satisfiability problem (MAX-SAT)

- Input: A boolean formula $\phi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$ in conjunctive normal form (CNF) .
- $M(\phi) = \{0,1\}^n$.
- $cost(\alpha)$ = # clauses of ϕ satisfied by α.
- goal: maximum

$$\phi = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge \bar{x}_2 \wedge (x_2 \vee x_3) \wedge x_3 \wedge (\bar{x}_1 \vee \bar{x}_3).$$

x_1	x_2	x_3	$x_1 \vee x_2$	$\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$	$\overline{x}_2$	$x_2 \vee x_3$	x_3	$\overline{x}_1 \vee \overline{x}_3$	# of satisfied clauses
0	0	0	0	1	1	0	0	1	3
0	0	1	0	1	1	1	1	1	5
0	1	0	1	1	0	1	0	1	4
0	1	1	1	1	0	1	1	1	5
1	0	0	1	1	1	0	0	1	4
1	0	1	1	1	1	1	1	0	5
1	1	0	1	1	0	1	0	1	4
1	1	1	1	0	0	1	1	0	3

Example: Integer linear programming (ILP)

- Input: An $m \times n$ matrix $A = [a_{ij}]_{i=1,\ldots m, j=1,\ldots,n}$ and two vectors b and c (integers entries).
- $M(A, b, c) = \{X | AX = b\}$.
- $cost(X) = c.X$.
- goal: minimum

Classification of Randomized Algorithms

for Optimization Problems

Definitions:

- A is a **consistent algorithm** for U if, for every $xinL$, the output $A(x)$ is a feasible solution for x (i.e., $A(x) \in M(x)$).
- Let A be a consistent algorithm for U. For every $x \in L$, we define the **approximation ratio** of A on x as

$$Ratio_A(x) = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}$$

- A is a δ-approximation algorithm for U if $Ratio_A(x) \leq \delta, \forall x \in L$.

Example: Approximation algorithm for vertex cover problem

Algorithm 4.1: VCA

Input: A graph $G(V, E)$.
Output: A vertex cover of $G = (V, E)$.

$C := \emptyset$; $A := \emptyset$ $E' := E$;
while $E' \neq \emptyset$ **do**
  take an arbitrary edge $(u, v) \in E'$;
  $C := C \cup \{u, v\}$; $A := A \cup \{u, v\}$;
  $E' := E' - \{$ all edges incident to u or $v\}$;
end
return C;

Example: Approximation algorithm for vertex cover problem

Claim: $Time_{VCA(G)} \in O(|E|)$: Obvious.

Claim: the algorithm VCA is a 2-approximation algorithm for MIN-VCP.

Proof.

- VCA is a consistent algorithm for MIN-VCP: Since $E' = \emptyset$ at the end of any computation, VCA computes a vertex cover in G.
- $|C| = 2\Delta|A|$.
- A is a matching in G.
- $Opt_{MIN-VCP}(G) \geq |A|$ (one needs $|A|$ vertives to cover matching A).

Hence, $\dfrac{|C|}{Opt_{MIN-VCP}(G)} = \dfrac{2|A|}{Opt_{MIN-VCP}(G)} \leq 2.$

Example: Approximation algorithm for vertex cover problem

Claim: $Time_{VCA(G)} \in O(|E|)$: Obvious.
Claim: the algorithm VCA is a 2-approximation algorithm for MIN-VCP.
Proof.

- VCA is a consistent algorithm for MIN-VCP: Since $E' = \emptyset$ at the end of any computation, VCA computes a vertex cover in G.
- $|C| = 2\Delta|A|$.
- A is a matching in G.
- $Opt_{MIN-VCP}(G) \geq |A|$ (one needs $|A|$ vertives to cover matching A).

Hence, $\dfrac{|C|}{Opt_{MIN-VCP}(G)} = \dfrac{2|A|}{Opt_{MIN-VCP}(G)} \leq 2.$

Main goal of randomization:

- Improve the approximation ratio.
- produce feasible solutions whose cost (quality) is not very far from oprimal cost.
  - to estimate the expected value $E(Ratio)$, or
  - to guarantee that an approximation ratio δ is achieved with probability at least $1/2$.

Classes of Ramdomized Algorithms:

- A randomized algorithm A is called a **randomized $E[\delta]$-approximation algorithm** for U if $\forall x \in L$
  (i) $Prob(A(x) \in M(x)) = 1$, and
  (ii) $E[Ratio_A(x)] \leq \delta$.
- For any positive real $\delta > 1$, a randomized algorithm A is called a **randomized δ-approximation algorithm** for U if $\forall x \in L$
  (i) $Prob(A(x) \in M(x)) = 1$, and
  (ii) $Prob(Ratio_A(x) \leq \delta) \geq 1/2$.

Claim:

These two classes are different in the strongly formal sense. i.e. a randomized $E[\delta]$-approximation algorithm for U is not necessarily a randomized δ- approximation algorithm for U, and vice versa.

Claim:

These two classes are different in the strongly formal sense. i.e. a randomized $E[\delta]$-approximation algorithm for U is not necessarily a randomized δ- approximation algorithm for U, and vice versa.

Proof.

Consider a randomized algorithm A that has exactly 12 runs, $C_1, C_2, \ldots, C_{12}$, on an input x, and all runs have the same probability and $Ratio_{A,x}(C_i) = 2$ for $i = 1, 2, \ldots, 10$ and $Ratio_{A,x}(C_j) = 50$ for $j \in \{11, 12\}$.

Wer have

- $E[Ratio_{A,x}] = \frac{1}{12}\Delta(10 \times 2 + 2 \times 50) = 10$
- $Prob(Ratio_{A,x} \leq 2) = 10 \times \frac{1}{12} = \frac{5}{6} \geq \frac{1}{2}$.

So, A is a randomized 2- approximation algorithm for U, but not a randomized $E[2]$-approximation algorithm.

Claim:

These two classes are different in the strongly formal sense. i.e. a randomized $E[\delta]$-approximation algorithm for U is not necessarily a randomized δ- approximation algorithm for U, and vice versa.

Proof. (Cont.)

Consider a randomized algorithm B that has exactly 1999 runs, on any input x, and all runs have the same probability. Assume that 1000 of these runs lead to results with approximation ratio 11 and that the remaining 999 runs compute an optimal solution. Wer have

- $E[Ratio_B] \sim 6$
- B is not a randomized δ-approximation algorithm for any $\delta < 11$.

Lemma 2.5.71.

Let $\delta > 0$ be a real number, and let U be an optimization problem. For any randomized algorithm B, if B is a randomized $E[\delta]$- approximation algorithm for U, then B is a randomized γ-approximation algorithm for U for $\gamma = 2 \times E[\delta]$.

Claim:

These two classes are different in the strongly formal sense. i.e. a randomized $E[\delta]$-approximation algorithm for U is not necessarily a randomized δ- approximation algorithm for U, and vice versa.

Proof. (Cont.)

Consider a randomized algorithm B that has exactly 1999 runs, on any input x, and all runs have the same probability. Assume that 1000 of these runs lead to results with approximation ratio 11 and that the remaining 999 runs compute an optimal solution. Wer have

- $E[Ratio_B] \sim 6$
- B is not a randomized δ-approximation algorithm for any $\delta < 11$.

Paradigms of the Design of Randomized Algorithms

- FOILING THE ADVERSARY
- ABUNDANCE OF WITNESSES
- FINGERPRINTING
- RANDOM SAMPLING
- AMPLIFICATION
- RANDOM ROUNDING