

CSL 857: Randomized Algorithms

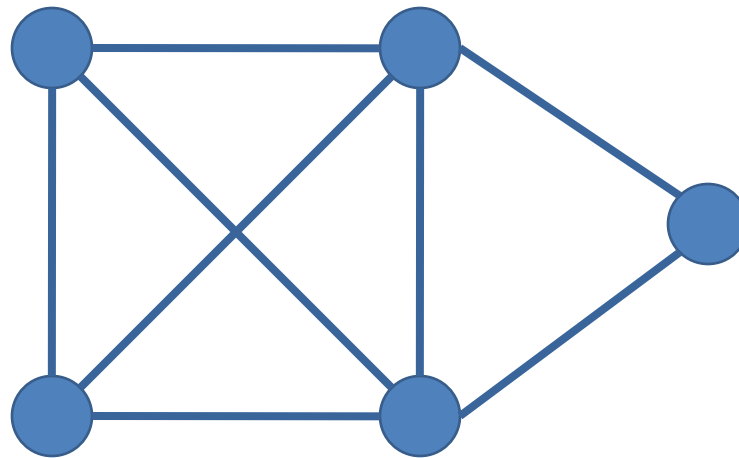
Ragesh Jaiswal
CSE. IIT Delhi

Randomized Algorithm

Karger's Min-Cut Algorithm

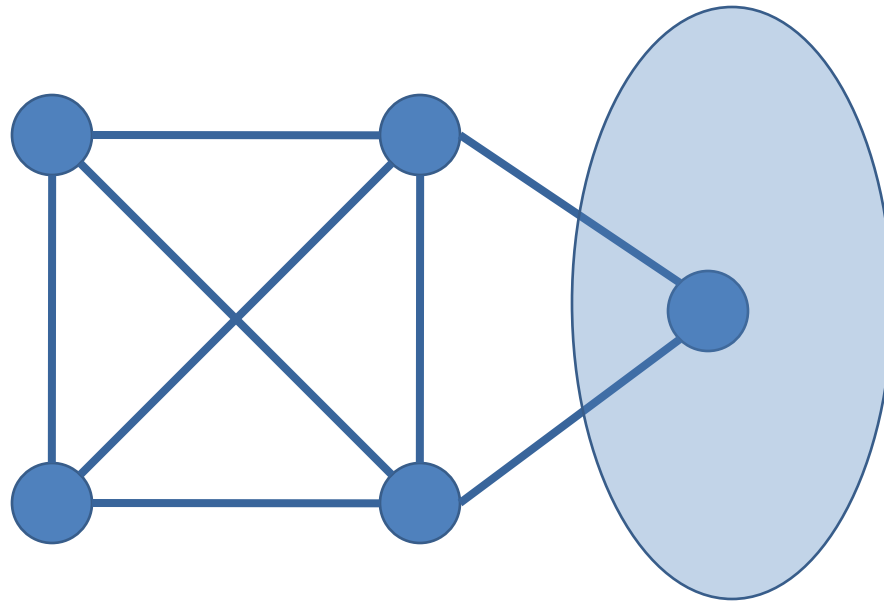
Karger's Min-Cut Algorithm

- Problem: Given a graph, find the cut of minimum size in the graph.
 - Cut: Partition of vertices into two non-empty sets. The edges between vertices in different sets are called cut edges. The size of the cut is the number of such edges.



Karger's Min-Cut Algorithm

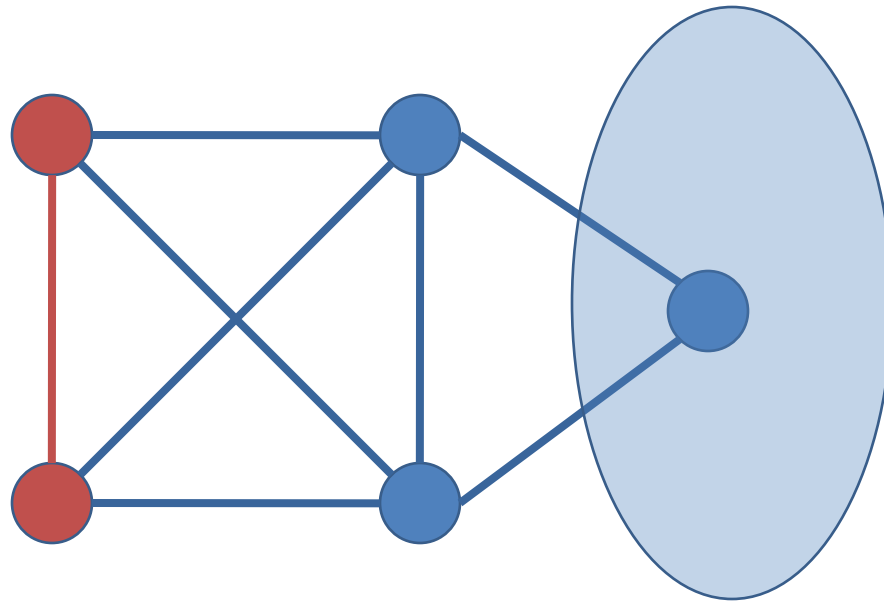
- Problem: Given a graph, find the cut of minimum size in the graph.
 - Cut: Partition of vertices into two non-empty sets. The edges between vertices in different sets are called cut edges. The size of the cut is the number of such edges.



Cut of minimum size

Karger's Min-Cut Algorithm

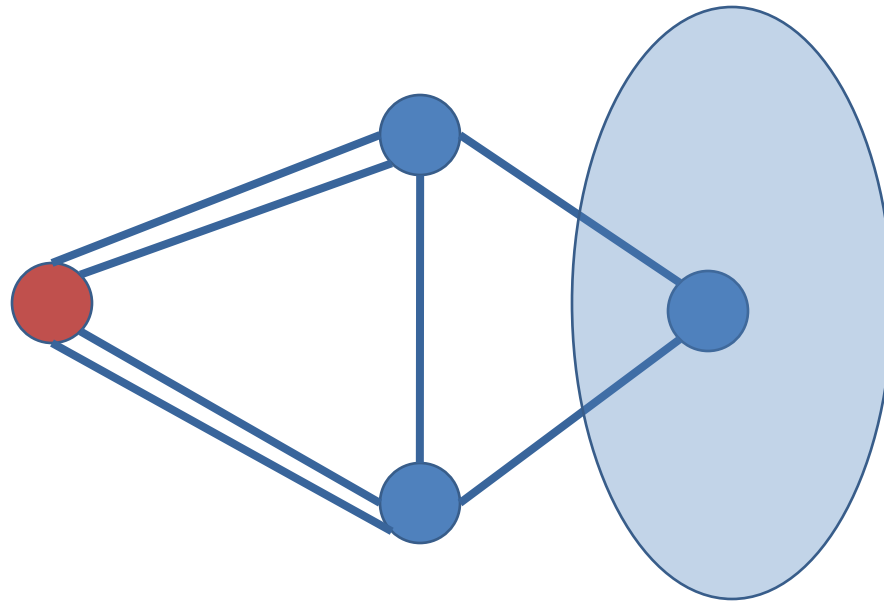
- *Collapse(e)*: Collapsing vertices across an edge $e=(u,v)$.
 - Merge vertices u and v and remove self loops. There may be multiple edges between same vertices (multi-graph).



Cut of minimum size

Karger's Min-Cut Algorithm

- *Collapse(e)*: Collapsing vertices across an edge $e=(u,v)$.
 - Merge vertices u and v and remove self loops. There may be multiple edges between same vertices (multi-graph).



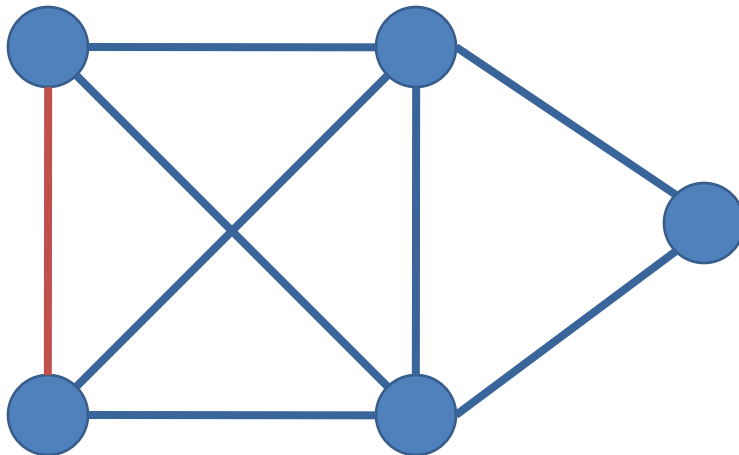
Cut of minimum size

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat $\mathbf{P}$ times // *For probability amplification.*
 - While $|\mathbf{V}| > 2$
 - Pick a random edge $\mathbf{e}$ in the multi-graph $\mathbf{G}$ and perform **Collapse**($\mathbf{e}$) to obtain $\mathbf{G}'$.
 - $\mathbf{G} \leftarrow \mathbf{G}'$
 - The edges across the remaining two vertices are the candidate cut edges.
 - Output the best answer from the $\mathbf{P}$ trials.

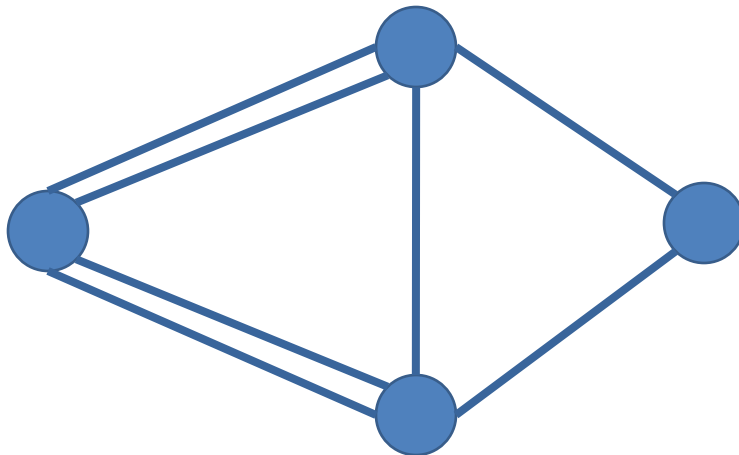
Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times *// For probability amplification.*
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.



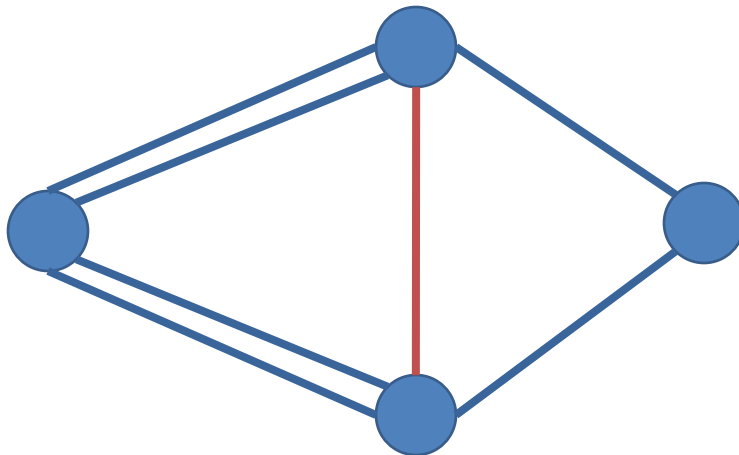
Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times *// For probability amplification.*
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.



Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times *// For probability amplification.*
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.



Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times // For probability amplification.
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.



Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times *// For probability amplification.*
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.



Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat $\mathbf{P}$ times *// For probability amplification.*
 - While $|\mathbf{V}| > 2$
 - Pick a random edge $\mathbf{e}$ in the multi-graph $\mathbf{G}$ and perform **Collapse**($\mathbf{e}$) to obtain $\mathbf{G}'$.
 - $\mathbf{G} \leftarrow \mathbf{G}'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the $\mathbf{P}$ trials.



The algorithm does well if the edges across a min-cut is never picked for collapsing.

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat $\mathbf{P}$ times *// For probability amplification.*
 - While $|\mathbf{V}| > 2$
 - Pick a random edge $\mathbf{e}$ in the multi-graph $\mathbf{G}$ and perform **Collapse**($\mathbf{e}$) to obtain $\mathbf{G}'$.
 - $\mathbf{G} \leftarrow \mathbf{G}'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the $\mathbf{P}$ trials.
- Let $\mathbf{C}$ denote the edges in a minimum size cut in $\mathbf{G}$.
- What is the probability that a randomly chosen edge belongs to $\mathbf{C}$?

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times // For probability amplification.
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.
- Let **C** denote the edges in a minimum size cut in **G**.
- What is the probability that a randomly chosen edge belongs to **C**?
 - Lemma 1: $\Pr[e \in C] \leq 2/n$.
 - Proof: Since $|E| \geq n |C| / 2$.

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat $\mathbf{P}$ times *// For probability amplification.*
 - While $|\mathbf{V}| > 2$
 - Pick a random edge $\mathbf{e}$ in the multi-graph $\mathbf{G}$ and perform **Collapse**($\mathbf{e}$) to obtain $\mathbf{G}'$.
 - $\mathbf{G} \leftarrow \mathbf{G}'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the $\mathbf{P}$ trials.
- Let $\mathbf{C}$ denote the edges in a minimum size cut in $\mathbf{G}$.
- Lemma 1: $\Pr[\mathbf{e} \in \mathbf{C}] \leq 2/n$.
- Lemma 2: If $\mathbf{e} \notin \mathbf{C}$, then size of min-cut of $\mathbf{G}'$ is the same as the size of min-cut of $\mathbf{G}$ after performing **Collapse**($\mathbf{e}$).

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat $\mathbf{P}$ times // For probability amplification.
 - While $|V| > 2$
 - Pick a random edge $\mathbf{e}$ in the multi-graph $\mathbf{G}$ and perform $\mathbf{Collapse}(\mathbf{e})$ to obtain $\mathbf{G}'$.
 - $\mathbf{G} \leftarrow \mathbf{G}'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the $\mathbf{P}$ trials.
- Let $\mathbf{C}$ denote the edges in a minimum size cut in $\mathbf{G}$.
- Lemma 1: $\Pr[\mathbf{e} \in \mathbf{C}] \leq 2/n$.
- Lemma 2: If $\mathbf{e} \notin \mathbf{C}$, then size of min-cut of $\mathbf{G}'$ is the same as the size of min-cut of $\mathbf{G}$ after performing $\mathbf{Collapse}(\mathbf{e})$.
- Let $\mathbf{D}_i$ denote the event that none of the edges in $\mathbf{C}$ have been used in the first i iterations.
- $\Pr[\mathbf{D}_{n-2}] = \prod \Pr[\mathbf{D}_{i+1} | \mathbf{D}_i] \geq (1-2/n)(1-2/n-1)\dots(1/3)$

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times // For probability amplification.
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.
- Let **C** denote the edges in a minimum size cut in **G**.
- Lemma 1: $\Pr[e \in C] \leq 2/n$.
- Lemma 2: If $e \notin C$, then size of min-cut of **G'** is the same as the size of min-cut of **G** after performing **Collapse(e)**.
- Let **D_i** denote the event that none of the edges in **C** have been used in the first **i** iterations.
- $$\begin{aligned}\Pr[D_{n-2}] &= \prod \Pr[D_{i+1} \mid D_i] \geq (1-2/n)(1-2/(n-1)) \dots (1/3) \\ &= 2/(n(n-1)) \\ &= \Omega(1/n^2)\end{aligned}$$

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat $\mathbf{P}$ times // *For probability amplification.*
 - While $|V| > 2$
 - Pick a random edge e in the multi-graph G and perform **Collapse**(e) to obtain G' .
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the $\mathbf{P}$ trials.
- $$\begin{aligned}\Pr[D_{n-2}] &= \prod \Pr[D_{i+1} \mid D_i] \geq (1-2/n)(1-2/n-1)\dots(1/3) \\ &= 2/(n(n-1)) \\ &= \Omega(1/n^2)\end{aligned}$$
- Let $\mathbf{P} = n^2(\ln n)$.
- What is the probability that the algorithm returns the correct answer?

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat **P** times // For probability amplification.
 - While $|V| > 2$
 - Pick a random edge **e** in the multi-graph **G** and perform **Collapse(e)** to obtain **G'**.
 - $G \leftarrow G'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the **P** trials.
- $\Pr[D_{n-2}] = \prod \Pr[D_{i+1} | D_i] \geq (1-2/n)(1-2/n-1)\dots(1/3)$
$$= 2/(n(n-1))$$
$$= \Omega(1/n^2)$$
- Let $P = n^2(\ln n)$.
- What is the probability that the algorithm returns the correct answer?
 - $\Pr[\text{Algorithm returns min-cut}] \geq 1 - (1 - 2/n^2)^{n^2 \ln n}$
$$\geq 1 - 1/n$$

Karger's Min-Cut Algorithm

- Min-cut algorithm:
 - Repeat $\mathbf{P}$ times // For probability amplification.
 - While $|V| > 2$
 - Pick a random edge $\mathbf{e}$ in the multi-graph $\mathbf{G}$ and perform $\mathbf{Collapse}(\mathbf{e})$ to obtain $\mathbf{G}'$.
 - $\mathbf{G} \leftarrow \mathbf{G}'$
 - The edges across the remaining two vertices are the cut edges.
 - Output the best answer from the $\mathbf{P}$ trials.
- Let $\mathbf{P} = \mathbf{n}^2(\ln \mathbf{n})$.
- $\Pr[\text{Algorithm returns min-cut}] \geq 1 - 1/\mathbf{n}$.
- Running time: $\mathbf{P} \cdot (\mathbf{n}-2) \cdot \mathbf{n} = \mathbf{O}(\mathbf{n}^4 \log \mathbf{n})$.
- Question: Is it possible to improve the running time?
 - Observation: The probability of a cut $\mathbf{C}$ surviving becomes smaller as the graph shrinks in size. So, “there is a need to repeat for smaller graphs not for larger ones”.

Karger's Min-Cut Algorithm

- Suppose we keep collapsing vertices until the number of vertices in the graph is $n/2$.
- What is the probability that a min-cut C has survived?
 - $\Pr[C \text{ survives}] \geq (1-2/n) \dots (1-2/(n/2+1)) \geq 1/4$.
- Starting from G we run the iterative collapse procedure four times independently to obtain graphs G_1, G_2, G_3 , and G_4 that have $n/2$ vertices. Repeat this step in a tree like fashion.
- Running time of above algorithm:
$$T(n) = 4T(n/2) + O(n^2) \Rightarrow T(n) = O(n^2 \log n)$$
- Success probability:

Karger's Min-Cut Algorithm

- Suppose we keep collapsing vertices until the number of vertices in the graph is $n/2$.
- What is the probability that a min-cut C has survived?
 - $\Pr[C \text{ survives}] \geq (1-2/n) \dots (1-2/(n/2+1)) \geq 1/4$.
- Starting from G we run the iterative collapse procedure four times independently to obtain graphs G_1, G_2, G_3 , and G_4 that have $n/2$ vertices. Repeat this step in a tree like fashion.
- Running time of above algorithm:
 $T(n) = 4T(n/2) + O(n^2) \Rightarrow T(n) = O(n^2 \log n)$
- Success probability: Let $P(n)$ denote the probability that a fixed min-cut C survives.
 - $P(n) \geq 1 - (1 - 1/4 \cdot P(n/2))^4$
 - $P(n) = \Omega(1/\log n)$

Karger's Min-Cut Algorithm

- Suppose we keep collapsing vertices until the number of vertices in the graph is $n/2$.
- What is the probability that a min-cut C has survived?
 - $\Pr[C \text{ survives}] \geq (1-2/n) \dots (1-2/(n/2+1)) \geq 1/4$.
- Starting from G we run the iterative collapse procedure four times independently to obtain graphs G_1, G_2, G_3 , and G_4 that have $n/2$ vertices. Repeat this step in a tree like fashion.
- Running time of above algorithm:
$$T(n) = 4T(n/2) + O(n^2) \quad \rightarrow \quad T(n) = O(n^2 \log n)$$
- Success probability: Let $P(n)$ denote the probability that a fixed min-cut C survives.
 - $P(n) \geq 1 - (1 - 1/4 \cdot P(n/2))^4$
 - $P(n) = \Omega(1/\log n)$
- So, we repeat the algorithm $O(\log^2 n)$ times to obtain C w.h.p.
- Overall running time: $O(n^2 \log^3 n)$.

Types of randomized algorithms

Monte Carlo Vs Las Vegas

Types of randomized algorithms

- We talked about two randomized algorithms:
 1. Randomized quick-sort
 2. Karger's min-cut algorithm

Types of randomized algorithms

- We talked about two randomized algorithms:
 1. Randomized quick-sort
 - The algorithm always produces correct answer.
 2. Karger's min-cut algorithm
 - The algorithm produces an incorrect answer with some small probability.

Types of randomized algorithms

- We talked about two randomized algorithms:
 1. Randomized quick-sort
 - The algorithm always produces correct answer.
 - Such randomized algorithms are called **Las Vegas** algorithms.
 2. Karger's min-cut algorithm
 - The algorithm produces an incorrect answer with some small probability.
 - Such randomized algorithms are called **Monte Carlo** algorithms.

Types of randomized algorithms

- We talked about two randomized algorithms:
 1. Randomized quick-sort
 - The algorithm always produces correct answer.
 - Such randomized algorithms are called **Las Vegas** algorithms.
 2. Karger's min-cut algorithm
 - The algorithm produces an incorrect answer with some small probability.
 - Such randomized algorithms are called **Monte Carlo** algorithms.
- Theorem (Monte-Carlo to Las Vegas): Given a Monte-Carlo algorithm for solving a problem that runs in expected time $T(n)$ and has a success probability of $\gamma(n)$. Further, given a solution, there is a deterministic algorithm can verify the correctness of the solution in time $t(n)$. Then there is a Las-Vegas algorithm for the problem that runs in expected time _____.

Types of randomized algorithms

- We talked about two randomized algorithms:
 1. Randomized quick-sort
 - The algorithm always produces correct answer.
 - Such randomized algorithms are called **Las Vegas** algorithms.
 2. Karger's min-cut algorithm
 - The algorithm produces an incorrect answer with some small probability.
 - Such randomized algorithms are called **Monte Carlo** algorithms.
- Theorem (Monte-Carlo to Las Vegas): Given a Monte-Carlo algorithm for solving a problem that runs in expected time $T(n)$ and has a success probability of $\gamma(n)$. Further, given a solution, there is a deterministic algorithm can verify the correctness of the solution in time $t(n)$. Then there is a Las-Vegas algorithm for the problem that runs in expected time $(T(n) + t(n))/\gamma(n)$.

Randomized complexity classes

RP, co-RP, BPP etc.

Complexity Classes

- Recap. of deterministic Turing Machines:

- Language: $L \subseteq \{0,1\}^*$

- Decision problem: Whether a given $x \in L$.

- Machine deciding a language: A Turing Machine M is said to decide a language if

$\forall x \in L, M(x) \text{ accepts}$

$\forall x \notin L, M(x) \text{ rejects}$

- Notion of efficiency (the class **P**): A Turing machine is said to be efficient if the running time of the machine is $\text{poly}(|x|)$ for some polynomial function poly .

Complexity Classes

- Recap. of deterministic Turing Machines:
 - Language: $L \subseteq \{0,1\}^*$
 - Decision problem: Whether a given $x \in L$.
 - Machine deciding a language: A Turing Machine M is said to decide a language if
$$\begin{aligned}\forall x \in L, M(x) &\text{ accepts} \\ \forall x \notin L, M(x) &\text{ rejects}\end{aligned}$$
 - Notion of efficiency (the class **P**): A Turing machine is said to be efficient if the running time of the machine is $\text{poly}(|x|)$ for some polynomial function poly .
- Randomized Turing Machine: A Turing Machine that uses extra *random tape* that has $r(n)$ random bits written on it.

Complexity Classes

- Randomized complexity class **RP**:
 - A language $L \subseteq \{0,1\}^*$ is said to be in the class **RP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq \frac{1}{2}, \text{ and}$$

$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] = 0.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$.

Complexity Classes

- Randomized complexity class **RP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class **RP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq \frac{1}{2}, \text{ and}$$
$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] = 0.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$.

- Let us define a class high-**RP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class high-**RP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq 1 - 2^{-k}, \text{ and}$$
$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] = 0.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$ and k is any large fixed constant (say $k = 100$).

Complexity Classes

- Randomized complexity class **RP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class **RP** if there is a randomized Turing machine **M** such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq \frac{1}{2}, \text{ and}$$
$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] = 0.$$

Moreover **M** runs in time $p(|x|)$ for some polynomial function $p(\cdot)$.

- Let us define a class high-**RP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class high-**RP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq 1 - 2^{-k}, \text{ and}$$
$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] = 0.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$ and k is any large fixed constant (say $k = 100$).

- Claim: **RP** = high-**RP**.

Complexity Classes

- Randomized complexity class **RP**:
 - A language $L \subseteq \{0,1\}^*$ is said to be in the class **RP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq \frac{1}{2}, \text{ and}$$
$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] = 0.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$.

- Claim 1: **RP** is closed under union.
- Claim 2: **RP** is closed under intersection.

Complexity Classes

- Relationship with class **P** and **NP**.
 - Recall **NP** contains all languages that have efficient *verifiers*. That is, a language $L \in NP$, if there is a TM **M** such that:
 $\forall x \in L, \exists y, s.t. |y| \leq p(|x|)$ and $M(x, y)$ accepts
 $\forall x \notin L, \forall y, s.t. |y| \leq p(|x|)$ and $M(x, y)$ rejects
Moreover, M runs in time $q(|x|)$. Here p and q are some polynomial functions.
 - Claim 1: $P \subseteq RP$

Complexity Classes

- Relationship with class **P** and **NP**.
 - Recall **NP** contains all languages that have efficient *verifiers*. That is, a language $L \in NP$, if there is a TM **M** such that:
 $\forall x \in L, \exists y, s. t. |y| \leq p(|x|)$ and $M(x, y)$ accepts
 $\forall x \notin L, \forall y, s. t. |y| \leq p(|x|)$ and $M(x, y)$ rejects
Moreover, M runs in time $q(|x|)$. Here p and q are some polynomial functions.
 - Claim 1: $P \subseteq RP$
 - Claim 2: $RP \subseteq NP$

Complexity Classes

- Randomized complexity class **BPP**:
 - A language $L \subseteq \{0,1\}^*$ is said to be in the class **BPP** if there is a randomized Turing machine M such that
$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq 3/4, \text{ and}$$
$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] < 1/4.$$
Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$.

Complexity Classes

- Randomized complexity class **BPP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class **BPP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq 3/4, \text{ and}$$

$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] < 1/4.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$.

- Let us define a class high-**BPP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class high-**BPP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq 1 - 2^{-k}, \text{ and}$$

$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] < 2^{-k}.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$ and k is any large fixed constant (say $k = 100$).

Complexity Classes

- Randomized complexity class **BPP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class **BPP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq 3/4, \text{ and}$$

$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] < 1/4.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$.

- Let us define a class high-**BPP**:

- A language $L \subseteq \{0,1\}^*$ is said to be in the class high-**BPP** if there is a randomized Turing machine M such that

$$\forall x \in L, \Pr[M(x, R) \text{ accepts}] \geq 1 - 2^{-k}, \text{ and}$$

$$\forall x \notin L, \Pr[M(x, R) \text{ accepts}] < 2^{-k}.$$

Moreover M runs in time $p(|x|)$ for some polynomial function $p(\cdot)$ and k is any large fixed constant (say $k = 100$).

- Claim: **BPP** = high-**BPP**

Complexity Classes

- Theorem (Chernoff bound): Let $n \geq 1$ be an integer and let $X_1, X_2, \dots, X_n$ be n independent 0/1 valued random variables. Let $X = X_1 + X_2 + \dots + X_n$ and let $a > 0$ be any real number. Then

$$\Pr[X \geq E[X] + an] \leq e^{-\frac{a^2 n}{2+a}}$$

$$\Pr[X \leq E[X] - an] \leq e^{-\frac{a^2 n}{2}}$$

End
