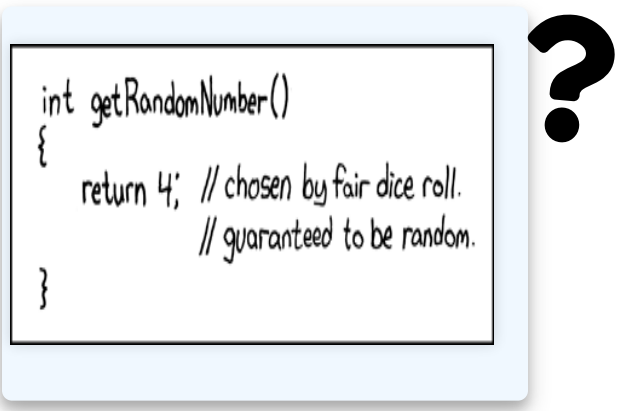
RANDOMIZED ALGORITHMS

PETER ROBINSON

peter.robinson@mcmaster.ca

WHY *RANDOMIZED* ALGORITHMS?

- Randomness is powerful resource for developing efficient algorithms with provable performance guarantees.
- Compared to deterministic algorithms, randomized algorithms are often...
  - faster (or use less memory),
  - simpler to understand, and...
  - easier to implement, e.g. fewer special cases to worry about.



Important to understand foundations!

WHAT THIS COURSE IS ABOUT

- How to use randomization to design better algorithms.
- Discuss many applications where access to randomness provides significant benefits.
- Equip you with necessary tools & techniques to analyze algorithms and also other random processes.
- Provide you with a foundation for using probabilistic concepts in your *own* work.

TOPICS (TENTATIVE)

- Concentration bounds
- Probabilistic data structures
- Fast graph algorithms
- Verification using fingerprinting techniques
- Random walks, Markov chains
- The probabilistic method
- Resilience against adversarial attacks in networks
- Symmetry breaking in networks
- Low-memory algorithms; dealing with dynamically-changing data

MARKING SCHEME (TENTATIVE *)

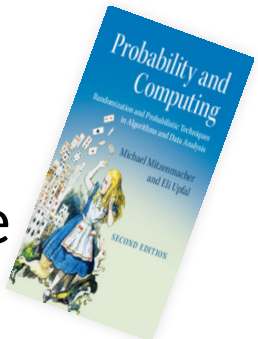
- Homework assignments (40%)
- Presentations of a research or survey paper (20%)
- Reviews of peers' presentations (5%)
- Final project (35%): choice of systems-focused or theory-focused

PREREQUISITES

- Knowledge of data structures & algorithms (undergrad level) is recommended... but most parts of the course are self-contained
- Undergrad-level discrete mathematics: discrete probability, basic knowledge of graph theory and combinatorics (e.g., how many ways can we choose an object such that...?)
- Basics of the Big-O notation (e.g., being able to figure out the meaning of $O(n \log n)$, $\Omega(n)$, etc.)
- Being able to write simple programs in `<insert your programming language of choice here>`
- Unsure if this course is suitable for you? Come talk to me.

RESOURCES

- Material is loosely based on: *Probability and computing: Randomized algorithms and probabilistic analysis* by Michael Mitzenmacher and Eli Upfal. 2nd edition, 2017. Cambridge University Press. (Not required but recommended.)



- Various resources on the web; to be added as we need them.

WHAT ARE *RANDOMIZED* ALGORITHMS?

Possible interpretations...

(1) Algorithms that make random choices during their executions. Use random number generator to decide next step.

(2) Algorithms that execute deterministically on randomly selected inputs.

Which interpretation is standard?

The answer is (1). Option (2) refers to *average case analysis* of algorithms.

ROADMAP FOR TODAY

- Verification of Matrix Multiplication
- Fast Min-Cut Computation
- Some techniques & tools along the way

PROBABILITY SPACE & PROBABILITY AXIOMS

Definition: A probability space $(\mathcal{S}, \mathcal{F}, \Pr)$ consists of...

- a sample space $\mathcal{S}$: set of all possible outcomes
- the set of events $\mathcal{F} \subseteq 2^{\mathcal{S}}$; for discrete prob. space $\mathcal{F} = 2^{\mathcal{S}}$.
- the probability function $\Pr : \mathcal{F} \to \mathbb{R}$

Definition: A probability function $\Pr$ satisfies

1. $\forall E \in \mathcal{F} : 0 \leqslant \Pr[E] \leqslant 1$
2. $\Pr[\mathcal{S}] = 1$
3. for all finite or countably infinite sequences of mutually disjoint events $E_1, E_2, \ldots$, it holds that $\Pr\left[\bigcup_{i \geqslant 1} E_i\right] = \sum_{i \geqslant 1} \Pr[E_i]$.

In discrete probability spaces: $\mathcal{S}$ either finite or countably infinite.

THE RAM MODEL

We analyze time complexity in the Random Access Machine model.

- Single processor, sequential execution
- Each simple operation takes 1 time step.
- Loops and subroutines are *not* simple operations.
- Each memory access takes one time step, and there is no shortage of memory.

APPLICATION: VERIFYING MATRIX MULTIPLICATION

THE PROBLEM

Input: Given three $n \times n$ matrices A, B, and C; entries are rational numbers.

Goal: Algorithm that verifies whether $A \cdot B = C$; answers either "yes" or "no"

First Attempt: "Verification by Computation"

Compute $(A \cdot B)$ and compare result with C.

Problem: standard matrix multiplication algorithm requires $O(n^3)$ time. More sophisticated algorithms still take $O(n^{2.3728639})$.

Can we solve this in $O(n^2)$ time if we avoid computing $A \cdot B$?

TOOL: SAMPLING UNIFORMLY AT RANDOM

Consider a finite set U of size m. We say that variable X is sampled uniformly at random **(u.a.r.)** from U when X is assigned to an element U, and each element in U has probability $\frac{1}{m}$ to be the chosen one.

Example - "Sampling a random bit":

$U = \{0, 1\}$. Variable X contains either 0 or 1 with equal probability.

Sampling 1 bit takes 1 unit of time in RAM model.

ALGORITHM FOR VERIFYING MATRIX MULTIPLICATION

Steps:	*Time:*
1. Sample n bits u.a.r. and store them in vector $\vec{r}$.	$O(n)$
2. Compute $\vec{x} := B \cdot \vec{r}$.	$O(n^2)$
3. Compute $\vec{y} := A \cdot \vec{x}$, so $\vec{y} = AB\vec{r}$.	$O(n^2)$
4. Compute $\vec{z} := C \cdot \vec{r}$	$O(n^2)$
5. if $\vec{y} = \vec{z}$ then output `yes`, else output `no`.	$O(n)$

Total running time:

$$O(n) + O(n^2) + O(n^2) + O(n^2) + O(n) = O(n^2).$$

Correctness?

CONDITIONAL PROBABILITY

Definition: Consider events E and F. The **conditional probability** $\Pr[E \mid F]$ is the probability that event E occurs given that F occurs:

$$\Pr[E \mid F] := \frac{\Pr[E \cap F]}{\Pr[F]}$$

Well defined only if $\Pr[F] > 0$.

Useful consequence:

$$\Pr[E \cap F] = \Pr[E \mid F] \Pr[F]$$

To simplify notation:

$$\Pr[A_1, \ldots, A_k] := \Pr\left[\bigcap_{i=1}^{k} A_i\right]$$

INDEPENDENCE OF EVENTS

Definition: We say events $A_1, \ldots, A_k$ are (mutually) independent if, for any $I \subseteq \{1, \ldots, k\}$:

$$\Pr\left[\bigcap_{i \in I} A_i\right] = \prod_{i \in I} \Pr[A_i]$$

We can simplify probability terms when conditioning on independent events:

$$\Pr[A_i \mid A_j] = \frac{\Pr[A_i \cap A_j]}{\Pr[A_j]} = \frac{\Pr[A_i]\Pr[A_j]}{\Pr[A_j]} = \Pr[A_i].$$

Do we require $i \neq j$?

Yes: A_i is clearly dependent on A_i.

TOOL: LAW OF TOTAL PROBABILITY

Theorem: Let $E_1, \ldots, E_k$ be mutually disjoint events that partition the sample space. Then, for any event B:

$$\Pr[B] = \sum_{i=1}^{k} \Pr[B \cap E_i] = \sum_{i=1}^{k} \Pr[B \mid E_i] \Pr[E_i]$$

Proof:

$$\begin{aligned}
\Pr[B] &= \Pr[B \cap (E_1 \cup \cdots \cup E_k)] \\
&= \Pr[(B \cap E_1) \cup (B \cap E_k)] \\
&= \sum_{i=1}^{k} \Pr[(B \cap E_i)] .
\end{aligned}$$

Now back to analyzing the algorithm...

CORRECTNESS OF ALGORITHM

- Deterministic algorithms either work or they don't.
- Not necessarily true for randomized algorithms!

Two things could go wrong:

1. Algorithm outputs `no` but correct answer is `yes`:

   Happens if $AB = C$ and $AB\vec{r} \neq C\vec{r}$.
   But, if $AB = C$, then also $AB\vec{r} = C\vec{r}$, for *any* $\vec{r}$. So, this can't happen here.

2. Algorithm outputs `yes` but correct answer is `no`:

   Happens if $AB \neq C$ and $AB\vec{r} = C\vec{r}$.
   This case is a bit trickier...

EXAMPLE

Let's look at instance where $AB \neq C$ and $AB\vec{r} = C\vec{r}$.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 3 & -2 \\ -1 & -2 & -1 \end{bmatrix}, B = \begin{bmatrix} 2 & 1 & -2 \\ -2 & 0 & 1 \\ 3 & -3 & 1 \end{bmatrix} C = \begin{bmatrix} 7 & -8 & 3 \\ -4 & 10 & -7 \\ 1 & 2 & -3 \end{bmatrix}$$

$$A \cdot B = \begin{bmatrix} 7 & -8 & 3 \\ -4 & 10 & -7 \\ -1 & 2 & -1 \end{bmatrix}$$

Will algorithm correctly output "no"?

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 3 & -2 \\ -1 & -2 & -1 \end{bmatrix}, B = \begin{bmatrix} 2 & 1 & -2 \\ -2 & 0 & 1 \\ 3 & -3 & 1 \end{bmatrix}$$
$$C = \begin{bmatrix} 7 & -8 & 3 \\ -4 & 10 & -7 \\ 1 & 2 & -3 \end{bmatrix}, A \cdot B = \begin{bmatrix} 7 & -8 & 3 \\ -4 & 10 & -7 \\ -1 & 2 & -1 \end{bmatrix}$$

Suppose we sample $\vec{r} = (0, 0, 1)^T$.

$A(B\vec{r}) = \begin{pmatrix} 3 \\ -7 \\ -1 \end{pmatrix}$ and $C\vec{r} = \begin{pmatrix} 3 \\ -7 \\ -3 \end{pmatrix}$. → Algorithm will output "no". Correct!

Now suppose we sample $\vec{r} = (1, 0, 1)^T$.

$A(B\vec{r}) = \begin{pmatrix} 10 \\ -11 \\ -2 \end{pmatrix}$ and $C\vec{r} = \begin{pmatrix} 10 \\ -11 \\ -2 \end{pmatrix}$. → Algorithm will output "yes". *Error!*

Algorithm detects $AB \neq C$ only for *"good"* choices of $\vec{r}$. Can we quantify probability of sampling good $\vec{r}$?

Lemma: If $AB \neq C$ and elements of $\vec{r}$ are chosen u.a.r., then $\Pr[AB\vec{r} = C\vec{r}] \leqslant \frac{1}{2}$.

Proof - High-level:

Define $D := AB - C$. Then $D \neq 0$.

Let $\vec{r} = (r_1, \ldots, r_n)^T$. Algorithm errs if $D\vec{r} = 0$.

For $D\vec{r} = 0$, it must be that $\sum_{j=1}^{n} (D_{1,j} \cdot r_j) = 0$.

▼ [Details]
We just focus on the **1st** row of D multiplied by $\vec{r}$. The same is true for all other rows of D too but it's not important for our purpose.

$$\Rightarrow r_1 = -\frac{1}{D_{1,1}} \cdot \sum_{j=2}^{n} (D_{1,j} \cdot r_j) \qquad (1)$$

Let's assume we sample bits in order $r_n, \ldots, r_2, r_1$.

After sampling $r_n, \ldots, r_2$: right-hand side of (1) is already fixed!

Since r_1 is u.a.r sampled from $\{0, 1\}$, probability that (1) holds $\leqslant \frac{1}{2}$.

Formal details of the Proof...

$$\Pr[AB\vec{r} = C\vec{r}] = \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr[(AB\vec{r} = C\vec{r}) \cap ((r_2,\ldots,r_n) = (x_2,\ldots,x_n))]$$

$$\leqslant \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left[\left(r_1 = -\frac{\sum_{j=2}^{n} D_{1,j} r_j}{D_{1,1}}\right) \cap ((r_2,\ldots,r_n) = (x_2,\ldots,x_n)\right]$$

$$= \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \Pr\left[(r_1 = -\frac{\sum_{j=2}^{n} D_{1,j} r_j}{D_{1,1}})\right] \cdot \Pr[(r_2,\ldots,r_n) = (x_2,\ldots,x_n))]$$

$$\leqslant \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \frac{1}{2} \cdot \Pr[(r_2,\ldots,r_n) = (x_2,\ldots,x_n))]$$

$$= \frac{1}{2}.$$

- We've seen that:
  (1) if $\boldsymbol{AB} \neq \boldsymbol{C}$, then algorithm is correct with probability $\geqslant \frac{1}{2}$;
  (2) if $\boldsymbol{AB} = \boldsymbol{C}$, then algorithm is correct with probability 1.
- This result doesn't seem very useful!
- Let's look at how to improve this.

BOOSTING THE PROBABILITY OF SUCCESS

1. Repeat the following k times:
   - Run randomized verification algorithm
   - If result is "no" break loop.
2. Output last result of randomized verification algorithm

$$\Rightarrow \Pr[\text{fails}] = \Pr[\text{ outputs "yes" in all } k \text{ trials}] \leqslant \left(\frac{1}{2}\right)^k.$$

- For $k = 50$, this is $\leqslant 8.88178 \times 10^{-16} \approx \frac{1}{1125899906842624}$.
- For $k = \lceil \log_2 n \rceil$, succeeds with high probability (w.h.p.), i.e.: $\geqslant 1 - \frac{1}{n}$.
- Works because error is one-sided: always correct if $AB = C$.

VERIFYING MATRIX MULTIPLICATION - WRAPPING UP

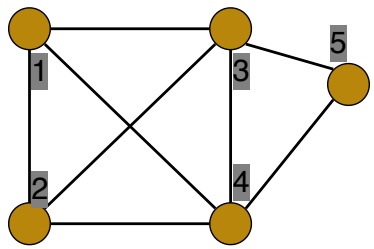
Theorem: There is a randomized algorithm for verifying matrix multiplication that runs in $O(n^2 \log n)$ time and succeeds with high probability.

Similar fingerprinting techniques have many applications (string equality verification, etc.)

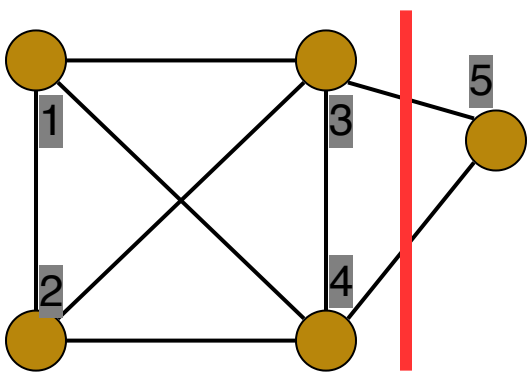
APPLICATION: FINDING THE MINIMUM CUT IN A GRAPH

THE MIN-CUT PROBLEM

- Consider connected undirected multigraph G with n vertices and m edges.



- A cut C of G is set of edges that disconnect G if we remove them.
- **Goal:** output the min-cut, which is a cut of minimum size.



$\{(3,5),(4,5)\}$

- Real-world applications: reliability of supply or computer networks

SIMPLE RANDOMIZED MIN-CUT ALGORITHM

Min-Cut Algorithm

Repeat until only 2 vertices left:

1. Sample edge (u, v) u.a.r. from available edges
2. Contract (u, v)
3. Remove self-loops but keep other multi-edges

Example

- $n = 5$
- Min-cut $\{(3, 5), (4, 5)\}$.

TOOL: CHAIN RULE OF CONDITIONAL PROBABILITY

Let $A_1, \ldots, A_k$ be not necessarily independent events. Then it holds:

$$\Pr[A_1, \ldots, A_k] = \Pr[A_1] \cdot \Pr[A_2 \mid A_1] \cdots \Pr[A_k \mid A_1, \ldots, A_{k-1}]$$

Proof: Inductively resolve conjunction of conditioned events

$$\begin{aligned}\Pr[A_1, \ldots A_k] &= \Pr[A_k \mid A_1, \ldots, A_{k-1}] \Pr[A_1, \ldots, A_{k-1}] \\ &= \Pr[A_k \mid A_1, \ldots, A_{k-1}] \Pr[A_{k-1} \mid A_1, \ldots, A_{k-2}] \Pr[A_1, \ldots, A_{k-2}] \\ &\ldots\end{aligned}$$

TOOL: UNION BOUND

Consider any events $A_1, \ldots, A_k$. Then

$$\Pr\left[\bigcup_{i=1}^{k} A_i\right] \leqslant \sum_{i=1}^{k} \Pr[A_i].$$

Proof:

Follows by induction.

To see intuition, just consider case $k = 2$:

$$\Pr[A_1 \cup A_2] = \Pr[A_1] + \Pr[A_2] - \Pr[A_1 \cap A_2]$$
$$\leqslant \Pr[A_1] + \Pr[A_2].$$

ANALYSIS OF MIN-CUT ALGORITHM

- Algorithm works as long as no min-cut edge is contracted
- Since the min-cut is small by definition, likely to succeed!
- What's the probability of sampling a min-cut edge?

Lemma: Let E_i be event that no min-cut edge is selected in step i. Let F_{i-1} be event that no min-cut edge was selected in steps $1, \ldots, i-1$. Then $\Pr[E_i \mid F_{i-1}] \geqslant 1 - \frac{2}{n-i+1}$.

Proof:

Conditioned on F_{i-1}, there are $n - (i-1) = n - i + 1$ vertices.

Suppose min-cut has size k. Then still $\geqslant \frac{k}{2}(n-i+1)$ edges left.

$\Rightarrow \Pr[\neg E_i \mid F_{i-1}] \leqslant \frac{2}{n-i+1}$.

$\Rightarrow \Pr[E_i \mid F_{i-1}] \geqslant 1 - \frac{2}{n-i+1}$.

Success probability is determined by $\Pr[F_{n-2}]$.

$$
\begin{aligned}
\Pr[F_{n-2}] &= \Pr[E_{n-2} \cap F_{n-3}] \\
&= \Pr[E_{n-2} \mid F_{n-3}] \Pr[F_{n-3}] \\
&= \Pr[E_{n-2} \mid F_{n-3}] \Pr[E_3 \mid F_{n-4}] \Pr[F_{n-4}] \\
&\geqslant \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) \\
&= \prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\left(\frac{n-5}{n-3}\right) \cdots \left(\frac{3}{5}\right)\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)}
\end{aligned}
$$

MIN-CUT: WRAPPING UP

Theorem: There is an algorithm that finds the min-cut with high probability in $O(n^4 \log n)$ time.

Proof:

1. Probability of Success:
   - Outputs is edge-set that is always some cut, i.e., one-sided error.
   - Repeat algorithm $\lceil n(n-1)\log n \rceil$ times and output smallest set found.
   - $\Pr[\text{fails}] \leqslant \left(1 - \frac{2}{n(n-1)}\right)^{n(n-1\log n)} \leqslant e^{-2\log n} = \frac{1}{n^2}.$

2. Time Complexity

- $n - 2$ iterations of sampling & contraction in base algorithm
- Sampling and contracting random edge takes $O(n)$
- We repeat base algorithm $O(n^2 \log n)$ times
- In total: $O(n^4 \log n)$ steps