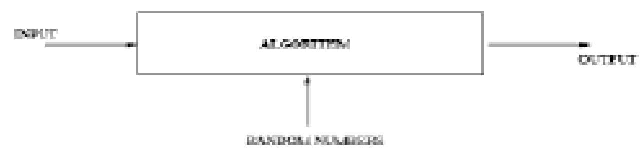
What is a randomized algorithm anyway?

Deterministic Algorithms

Goal: To prove that the algorithm solves the problem *correctly* (always) and *quickly* (typically, the number of steps should be polynomial in the size of the input).

Randomized Algorithms



- In addition to input, algorithm takes a source of random numbers and makes random choices during execution.
- Behavior can vary even on a fixed input.

Why randomize?
- Speed
- Adversarial inputs
- Simplicity

Applications of Randomized Algorithms

- **Number-theoretic algorithms:** Primality testing (Monte Carlo).
- **Data structures:** Sorting, order statistics, searching, computational geometry.
- **Algebraic identities:** Polynomial and matrix identity verification. Interactive proof systems.
- **Parallel and distributed computing:** Deadlock avoidance, distributed consensus.
- **Probabilistic existence proofs:** Show that a combinatorial object arises with non-zero probability among objects drawn from a suitable probability space.
- **Mathematical programming:** Faster algorithms for linear programming. Rounding linear program solutions to integer program solutions.
- **Graph algorithms:** Minimum spanning trees, shortest paths, minimum cuts.
- **Counting and enumeration:** Matrix permanent. Counting combinatorial structures.
- **Derandomization:** First devise a randomized algorithm, then argue that it can be "derandomized to yield a deterministic algorithm.

Tools for Randomized Algorithms

- Basic Probability Tools: Random Variables, Expectation, Conditional Expectation.
- Intermediate Probability Tools: Moments, Deviation (Markov & Chebyshev Inequalities)
- Game Theoretic Techniques (Highlight: Yao's Minimax Principle)
- Tail Inequalities (Highlights: Chernoff Bounds, Martingales)
- Probabilistic Method (Highlights: Sample & Modify, Lovasz Local Lemma)
- Markov Chains & Random walks (Highlights: Cover Time, Role of Expanders)
- Markov Chain Monte Carlo Method (Metropolis Algorithm, Approximate Counting)
- Algebraic Techniques (Fingerprinting Methods, Interactive Proof Systems & PCP Theorem)
- Miscellaneous (Derandomization, Randomized Rounding)
- Post-2000 (Statistical Physics, Phase transitions, Replica Method etc.)

But this is only the tip of the iceberg – the real story is as they say stranger than fiction and is only beginning to unfold. Lot's of opportunities for young guns.

Monte Carlo and Las Vegas

Monte Carlo and Las Vegas

A **Monte Carlo** algorithm runs for a fixed number of steps, and produces an answer that is correct with probability $\geq 1/3$.

A **Las Vegas** algorithm always produces the correct answer; its running time is a random variable whose expectation is bounded (say by a polynomial).

Monte Carlo and Las Vegas

These probabilities/expectations are only over the random choices made by the algorithm – *independent of the input*.

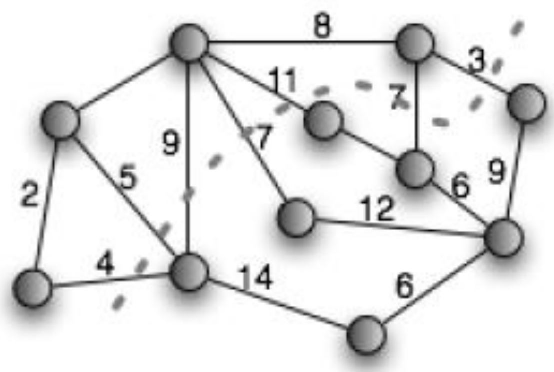
Thus independent repetitions of Monte Carlo algorithms drive down the failure probability exponentially.

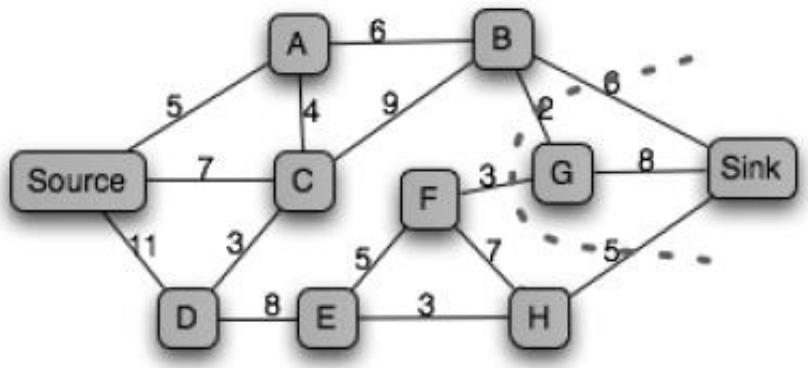
Comparison Chart

Las Vegas	Monte Carlo
Output: Always correct.	Output: Probabilistically correct. Correct with probability p. Incorrect with probability 1-p
Time: Running time is a Random Number with bounded expectation	Time: Fixed number of steps. Usually polynomial time.

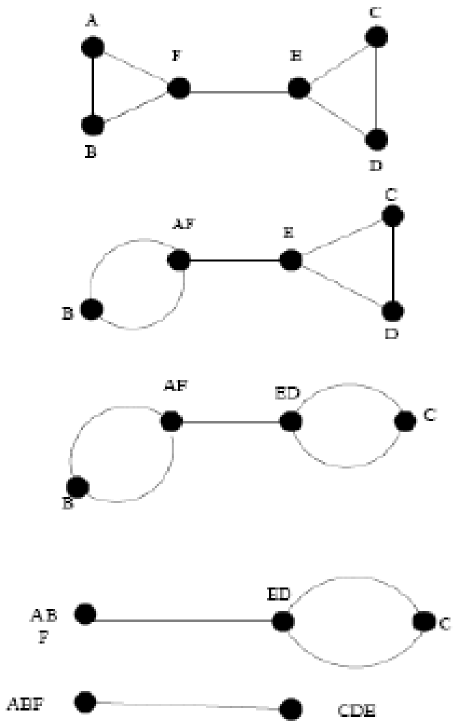
Min-Cut

- In graph theory, a cut is a partition of the vertices of a graph into two disjoint subsets.
- The cut-set of the cut is the set of edges whose end points are in different subsets of the partition. Edges are said to be crossing the cut if they are in its cut-set.





Randomized min-cut



RANDOMIZED MIN-CUT(G)

1: **for** $i = 1$ **to** $n - 2$ **do**
2: Pick an edge e_i in G uniformly at random
3: Contract two end points of e_i (remove loops)
4: **end for**
5: // At this point, two vertices u, v left
6: Output all remaining edges between u and v

Does the algorithm terminate in polynomial time?

What is the probability that this algorithm discovers a min-cut ?!

- Let C be a minimum cut, $k = |C|$
- If no edge in C is chosen by the algorithm, then C will be returned in the end, and vice versa
- For $i = 1..n-2$, let A_i be the event that $e_i \notin C$ and B_i be the event that $\{e_1, \ldots, e_i\} \cap C = \emptyset$

A_i and B_i are intermediate probabilities that are telling you that everything is going according to the plan… "A" variables are per step information. "B" variables are cumulative. B_1 = ? B_{n-2} = ?. A_i can be conditioned on B_{i-1}

All actors are cast

- Let C be a minimum cut, $k = |C|$
- If no edge in C is chosen by the algorithm, then C will be returned in the end, and vice versa
- For $i = 1..n-2$, let A_i be the event that $e_i \notin C$ and B_i be the event that $\{e_1, \ldots, e_i\} \cap C = \emptyset$

$$
\begin{aligned}
& \text{Prob}[C \text{ is returned}] \\
= \; & \text{Prob}[B_{n-2}] \\
= \; & \text{Prob}[A_{n-2} \cap B_{n-3}] \\
= \; & \text{Prob}[A_{n-2} \mid B_{n-3}]\,\text{Prob}[B_{n-3}] \\
= \; & \ldots \\
= \; & \text{Prob}[A_{n-2} \mid B_{n-3}]\,\text{Prob}[A_{n-3} \mid B_{n-4}] \cdots \text{Prob}[A_2 \mid B_1]\,\text{Prob}[B_1]
\end{aligned}
$$

Now we need to go forward !!

Processing...

- At step 1, G has min-degree $\geq k$, hence $\geq kn/2$ edges

$$\text{Prob}[B_1] = \text{Prob}[A_1] \geq 1 - \frac{k}{kn/2} = 1 - \frac{2}{n}$$

- Now we estimate $\text{Prob}[A_2 \mid B_1]$.
  - At step 2, the min cut is still at least k, hence $\geq k(n-1)/2$ edges
  - Thus, similar to step 1 we have

$$\text{Prob}[A_2 \mid B_1] \geq 1 - \frac{2}{n-1}$$

- In general,

$$\text{Prob}[A_j \mid B_{j-1}] \geq 1 - \frac{2}{n-j+1}$$

QED page

$$\text{Prob}[C \text{ is returned}]$$

$$= \quad \text{Prob}[A_{n-2} \mid B_{n-3}]\,\text{Prob}[A_{n-3} \mid B_{n-4}] \cdots \text{Prob}[A_2 \mid B_1]\,\text{Prob}[B_1]$$

Meanwhile

$$\text{Prob}[B_1] = \text{Prob}[A_1] \geq 1 - \frac{k}{kn/2} = 1 - \frac{2}{n}$$

$$\text{Prob}[A_j \mid B_{j-1}] \geq 1 - \frac{2}{n-j+1}$$

$$\text{Prob}[C \text{ is returned}] \geq \prod_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)}$$

Last step is homework exercise

How do you lower failure probability

- The basic algorithm has failure probability at most $1 - \frac{2}{n(n-1)}$
- How do we lower it?
- Run the algorithm multiple times, say $m \cdot n(n-1)/2$ times, return the smallest cut found
- The failure probability is at most

$$\left(1 - \frac{2}{n(n-1)}\right)^{m \cdot n(n-1)/2} < \frac{1}{e^m}.$$

Last step is homework exercise.

RANDOMIZED-QUICKSORT(A)

randomization

$n \leftarrow$ length(A)
if $n = 1$ **then**
    Return A
else
    Pick $i \in \{1, \ldots, n\}$ uniformly at random. $A[i]$ is called the *pivo*
    $L \leftarrow$ elements $\leq A[i]$
    $R \leftarrow$ elements $> A[i]$
    // the above takes one pass through A
    $L \leftarrow$ RANDOMIZED-QUICKSORT(L)
    $R \leftarrow$ RANDOMIZED-QUICKSORT(R)
    Return $L \cdot A[i] \cdot R$
end if

- The running time is proportional to the number of comparisons
- Let $b_1 \leq b_2 \leq \cdots \leq b_n$ be A sorted non-decreasingly
- For each $i < j$, let X_{ij} be the indicator random variable indicating if b_i was ever compared with b_j
- The expected number of comparisons is

$$\mathsf{E}\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} \mathsf{E}[X_{ij}] = \sum_{i<j} \text{Prob}[b_i \ \& \ b_j \text{ was compared}]$$

- b_i was compared with b_j if and only if either b_i or b_j was chosen as a pivot before any other in the set $\{b_i, b_{i+1}, \ldots, b_j\}$
- Hence, $\text{Prob}[b_i \ \& \ b_j \text{ was compared}] = \frac{2}{j-i+1}$

$$
\begin{aligned}
E[C] &= \sum_{i=1}^{n} \sum_{j>i} E[C_{i,j}] \\
&= \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&\leq 2n \left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right) \\
&= O(n \log n)
\end{aligned}
$$

Harmonic number acts as a proxy to log(n)