ROMA TRE
UNIVERSITÀ DEGLI STUDI

Tecniche Algoritmiche per Grafi e Reti

Algoritmi randomizzati

Patrizio Angelini

Dipartimento di Informatica e Automazione
Università degli Studi Roma Tre

Riferimenti

Probabilistic Methods in CS

Eli Upfal
Brown University
eli@cs.brown.edu

Randomized Algorithms

Prabhakar Raghavan
IBM Almaden Research Center
San Jose, CA.

Verifying Polynomial Identities

Problem: Verify that $P(x) \equiv Q(x)$.
Example: Check if

$$(x+1)(x-2)(x+3)(x-4)(x+5)(x-6) \equiv x^6 - 7x^3 + 25.$$

(We use $\equiv$ for polynomial identities, $=$ for numerical equality.)

Deterministic solution:

$$\begin{aligned} H(x) &\equiv (x+1)(x-2)(x+3)(x-4)(x+5)(x-6) \\ G(x) &\equiv x^6 - 7x^3 + 25. \end{aligned}$$

Transform $H(x)$ to a "canonical" form

$$H(x) \equiv \sum_{i=0}^{6} c_i x^i$$

$H(x) \equiv G(x)$ iff the coefficients of all monomials are equal.

A randomized solution

"Devo verificare un'equivalenza polinomiale…Che fò? Fò dù conti"

(Prof. G. Di Battista)

A randomized solution

- Choose a random integer r in the range [1,…,600]
- Compute $H(r)$ and $G(r)$
- If $H(r) = G(r)$ then output TRUE, otherwise output FALSE

Randomized Algorithm

$$H(x) \equiv (x+1)(x-2)(x+3)(x-4)(x+5)(x-6)$$
$$G(x) \equiv x^6 - 7x^3 + 25.$$

Assume $r = 2$

$H(2) = 3 \times 0 \times 5 \times -2 \times 7 \times -4 = 0.$

$G(2) = 2^6 - 72^3 + 25 = 64 - 56 + 25 = 33.$

Since $H(2) \neq G(2)$ we proved that $H(x) \not\equiv G(x)$.

What happens if we have equality?

Example 1: Check if $(x+1)(x-1) \equiv x^2 - 1$.
Since the two sides of the equation are identical - any number that we try would give equality.

Randomized Algorithm

Example 2: Check if $x^2 + 7x + 1 = (x+2)^2$.
If we try $r = 2$ we get

$$LHS = 4 + 14 + 1 = 19, \qquad RHS = 4^2 = 16$$

showing that the two sides are not identical. But for $r = 1$ we get

equality:

$$1 + 7 + 1 = (1+2)^2 = 9.$$

A bad choice of r may lead to a wrong answer!

Some Algebra

$F(x) \equiv G(x) - H(x)$ is a polynomial in one variable of degree bounded by d.

Theorem

If

$$F(x) = G(x) - H(x) \not\equiv 0$$

then the equation

$$F(x) = G(x) - H(x) = 0$$

has no more than d roots (solutions).

Analysis of the Algorithm

If the identity is correct, the algorithm always outputs a correct answer.
If the identity is NOT correct, the algorithm outputs the WRONG answer only if we randomly picked r which is a root of the polynomial $F(x) = G(x) - H(x) = 0$.
If we choose r in the range $[1, ..., 100d]$, the "chance" of returning a wrong answer is no more than 1%.

A randomized technique gives a significantly simpler algorithm - at a cost of a small probability of error.

Getting an arbitrary small error probability

We can reduce the "error probability" at the expense of increasing the run-time of the algorithm:

1. Run the algorithm 10 times.
2. Output "CORRECT" if got "CORRECT" in all the 10 runs.

If the new algorithm outputs "CORRECT" The "chance" that $G(x) \neq H(x)$ is less than $10^{-20} < 2^{-64}$.

The general Result

Theorem

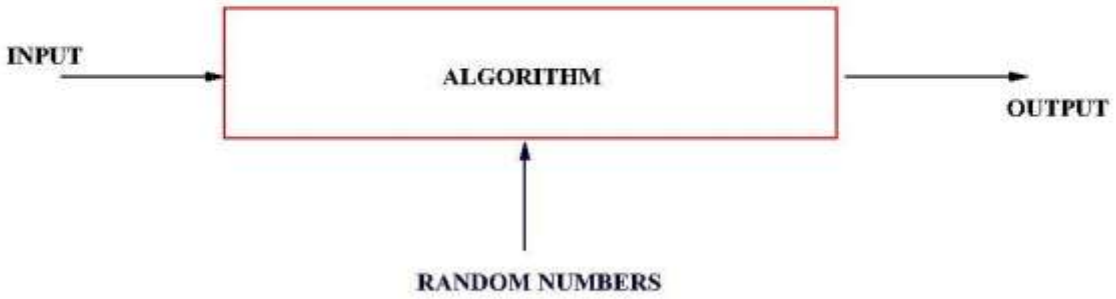
Let $f(X_1, \ldots, X_n)$ be a multivariant polynomial of degree d. If r_i, $i = 1, \ldots, n$, is chosen uniformly at random from $[0, \ldots, 2d]$ then

$$\Pr(f(r_1, \ldots, r_n) = 0 \mid f(X_1, \ldots, X_n) \not\equiv 0) \leq 1/2.$$

Deterministic Algorithms

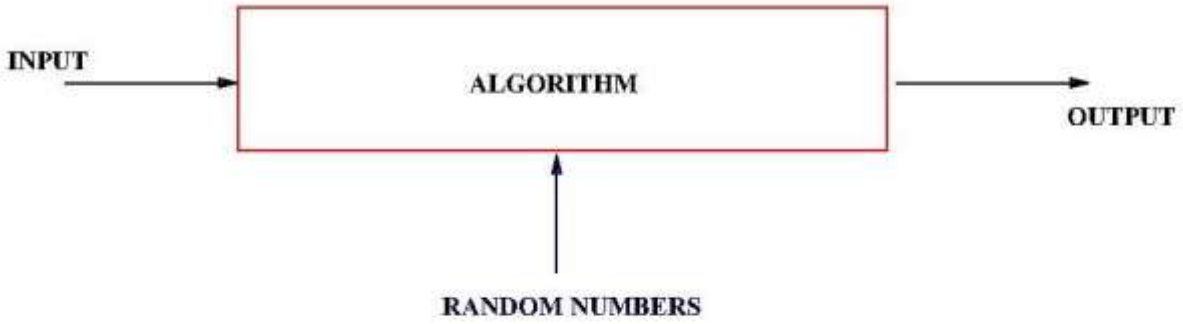
Goal: To prove that the algorithm solves the problem *correctly* (always) and *quickly* (typically, the number of steps should be polynomial in the size of the input).

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.

Randomized Algorithms



- Design algorithm + analysis to show that this behavior is likely to be good, on *every input*.
  (The likelihood is over the random numbers only.)

Not to be confused with the Probabilistic Analysis of Algorithms

- Here the *input* is assumed to be from a probability distribution.
- Show that the algorithm works for most inputs.

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 and its *running time* is a *random variable* whose expectation is *bounded* (usually by a polynomial)

Monte Carlo and Las Vegas

- These probabilites and expectations are only *over the random choices* made by the algorithm
  - *Independent on the input*
- Thus, *independent repetitions* of a Monte Carlo algorithm make the failure probability drop down *exponentially*
  - *Chernoff Bound*

Monte Carlo → Las Vegas

- A Monte Carlo algorithm can be turned into a Las Vegas algorithm if an *efficient procedure* exists to *check* whether an answer is *correct*
  - the Monte Carlo algorithm is run repeatedly till a correct answer is obtained.

Las Vegas → Monte Carlo

- A Las Vegas algorithm can be always turned into a Monte Carlo algorithm
  - Repeat it for a fixed amount of time

Scope

- Number-theoretic algorithms
  - Primality testing (Monte Carlo)
- Data Structures
  - Sorting and searching
- Algebraic identities
  - Polynomial and matrix identities verification
- Mathematical programming
  - Faster linear programming algorithms, rounding linear programming to integer programming solutions

Scope

- Graph algorithms
  - Minimum spanning trees, shortest paths, min-cut
- Parallel and distributed computing
  - Deadlock avoidance, distributed consensus
- Probabilistic existence proofs
  - Show that a combinatorial object arises with a positive probability among objects drawn from a siutably probability space

Randomized Algorithms: Advantages

- Simplicity
- Performances

For many problems, a randomized algorithm is the simplest, the fastest, or both

Randomized Algorithms: Advantages

"In *testing primality* of very large numbers chosen at *random*, the *chance* of stumbling upon a value that *fools the Fermat test* is *less than the chance* that *cosmic radiation* will cause the computer to make an error in carrying out a *'correct'* algorithm. Considering an algorithm to be inadequate for the first reason but not for the second illustrates the *difference* between *mathematics* and *engineering*."

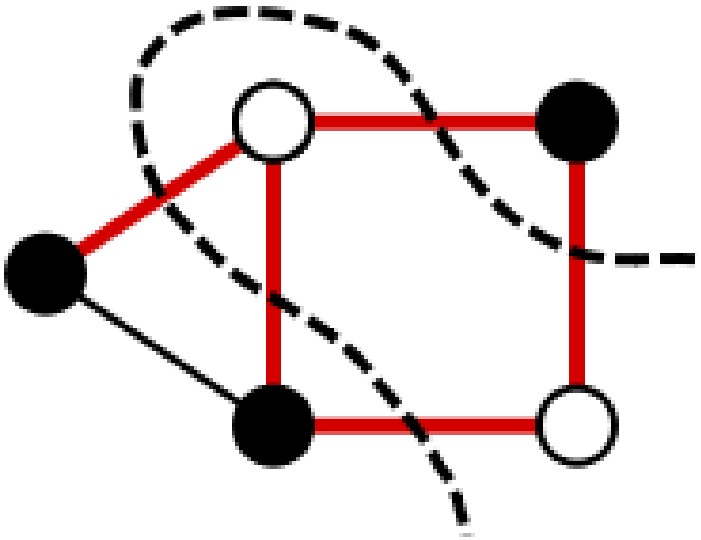
- Han Abelson and Gerald J. Sussman (1996). *Structure and Interpretation of Computer Programs. MIT Press.*

Min-Cut Problem

- 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.
- The weight of a cut is the sum of the weights of the edges crossing the cut. If the graph is unweighted, each edge has weight 1.

Min-Cut Problem

- A *cut* is a partition of the vertices of a graph into two disjoint subsets.



Weight = 5

Min-Cut Problem

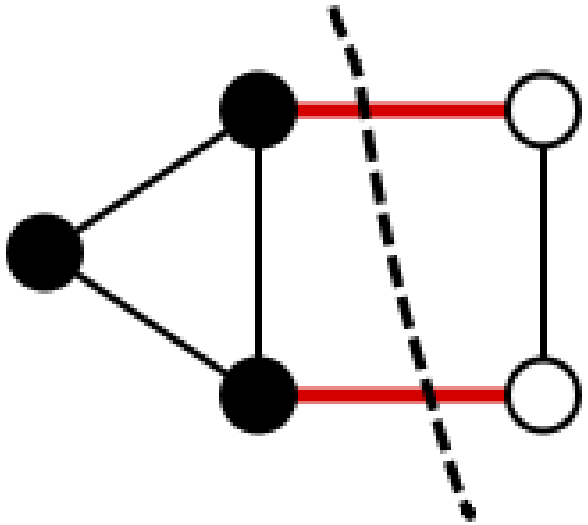
- A cut is *minimum* if the size of the cut is not larger than the size of any other cut

- The maximum flow of the network and the weight of any minimum cut are equal
  - Max-flow Min-cut theorem

- Min-cut problem is polynomial-time solvable
  - Ford-Fulkerson $O(m*f)$
  - Edmonds-Karp $O(n*m^2)$
  - Stoer and Wagner (1994) $O(nm+n^2 \log n)$

Min-Cut Problem

- A cut is *minimum* if the size of the cut is not larger than the size of any other cut



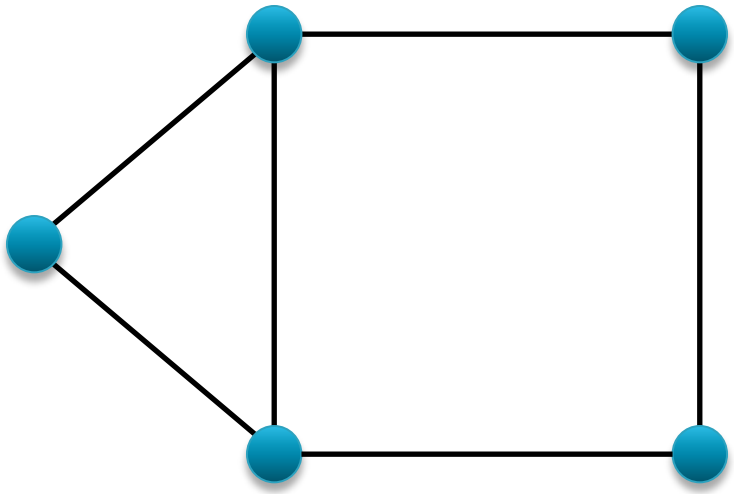
Weight = 2

Min-Cut Randomized Algorithm

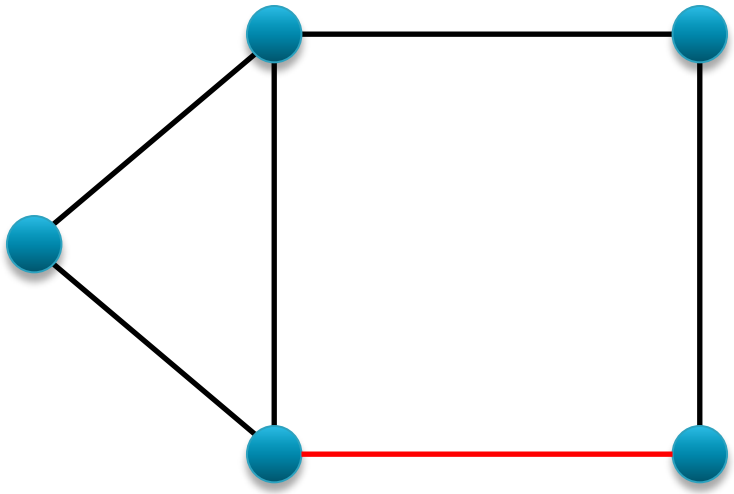
- **Input:** An n-node graph G.
- **Output:** A minimal set of edges that disconnects the graph.

1. Repeat n − 2 times:
   a. Pick an edge uniformly at random.
   b. Contract the two vertices connected by that edge and remove all the edges connecting them.
2. Output the set of edges connecting the two remaining vertices.

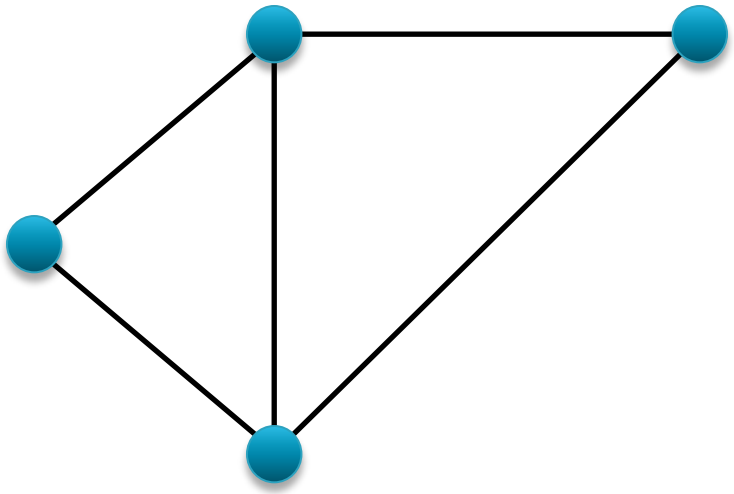
Min-Cut Randomized Algorithm



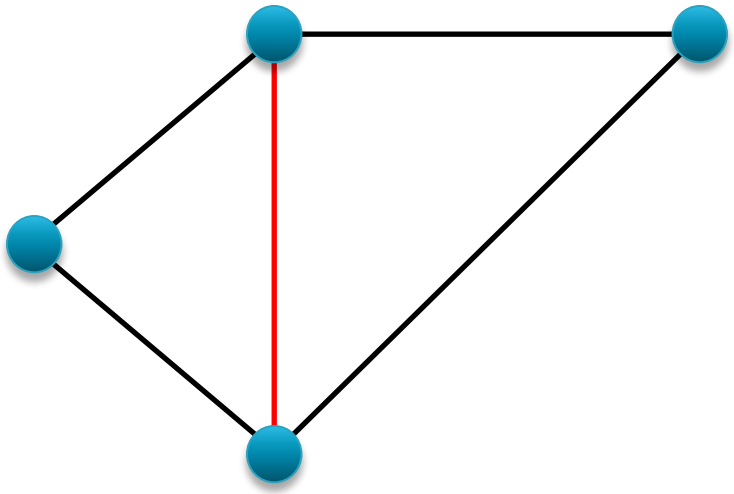
Min-Cut Randomized Algorithm



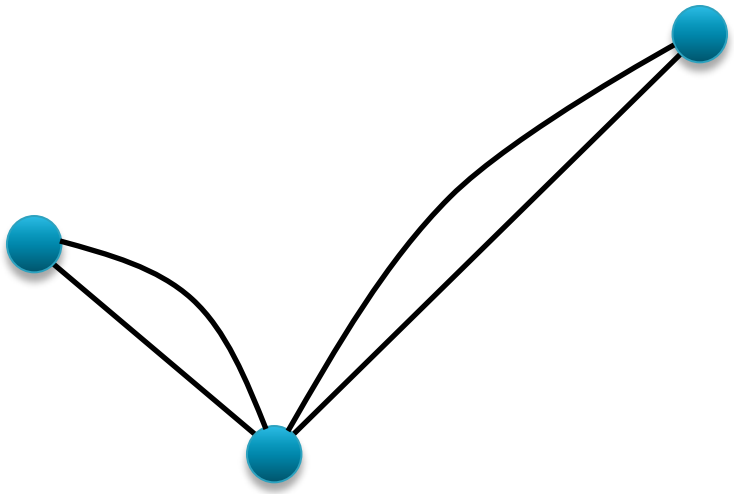
Min-Cut Randomized Algorithm



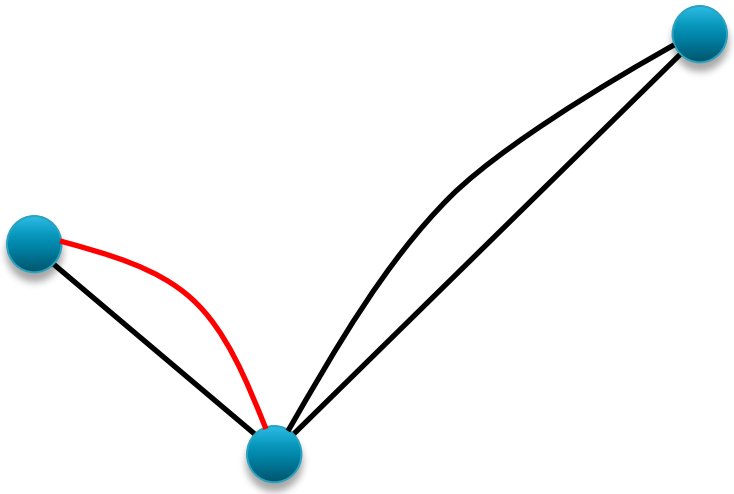
Min-Cut Randomized Algorithm



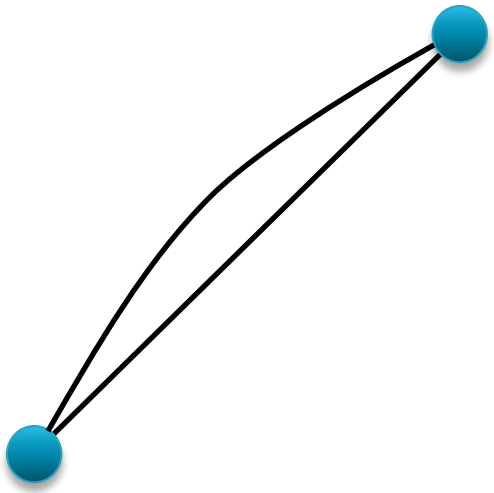
Min-Cut Randomized Algorithm



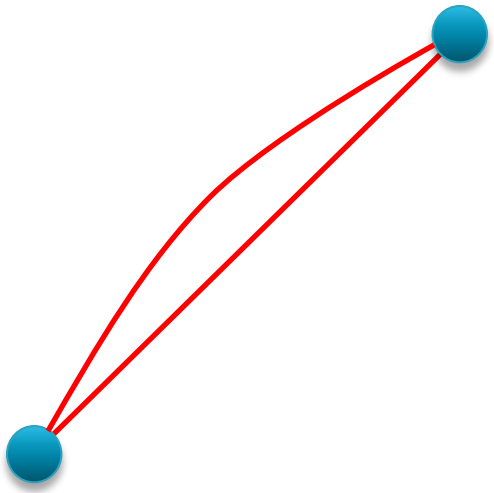
Min-Cut Randomized Algorithm



Min-Cut Randomized Algorithm

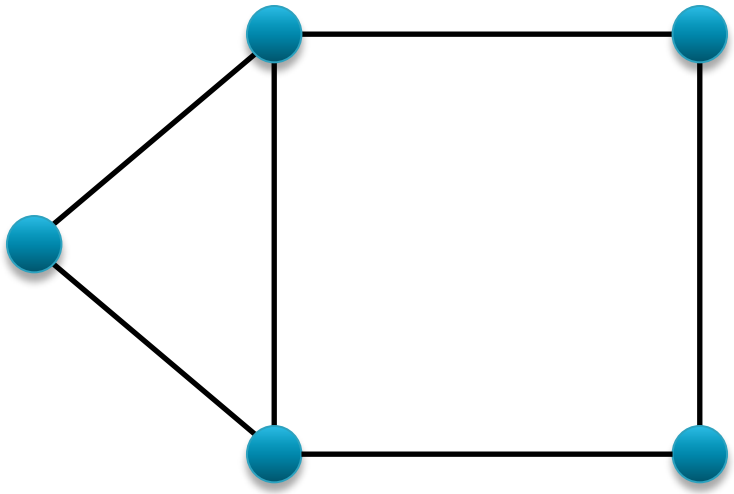


Min-Cut Randomized Algorithm

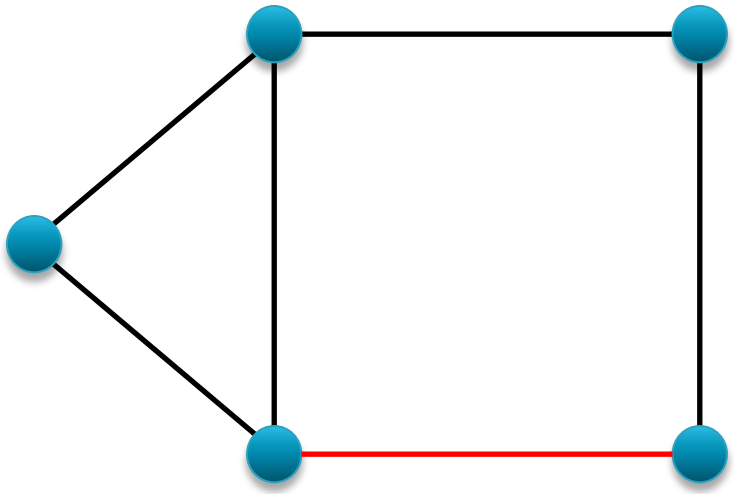


Weight = 2

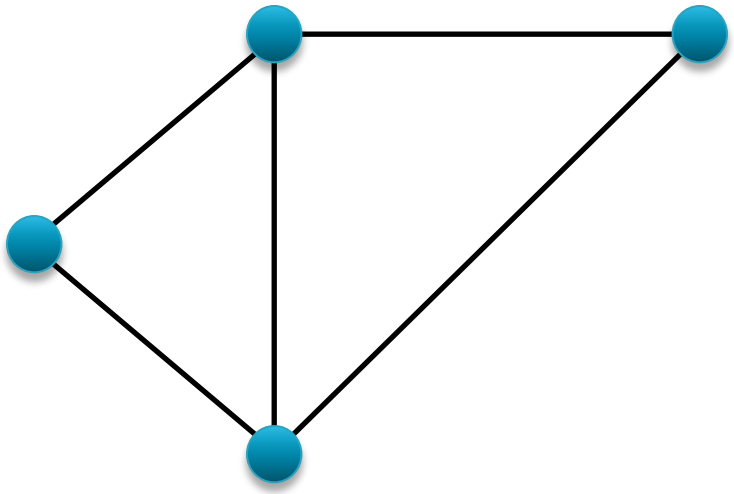
Min-Cut Randomized Algorithm



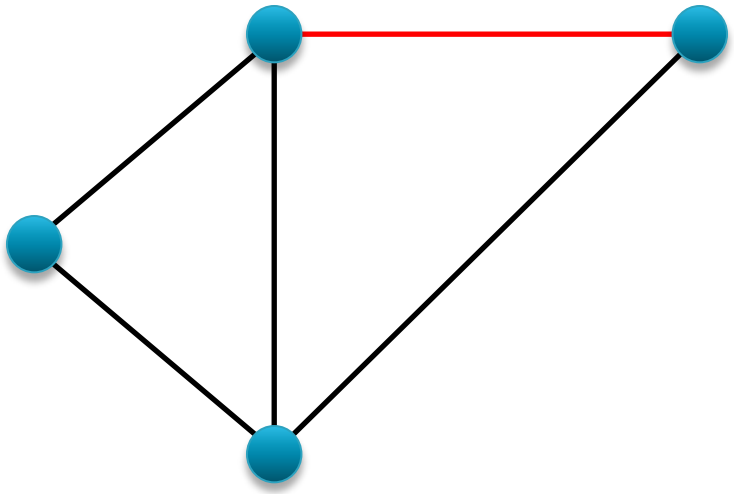
Min-Cut Randomized Algorithm



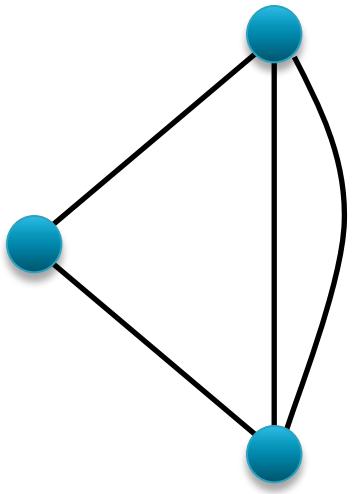
Min-Cut Randomized Algorithm



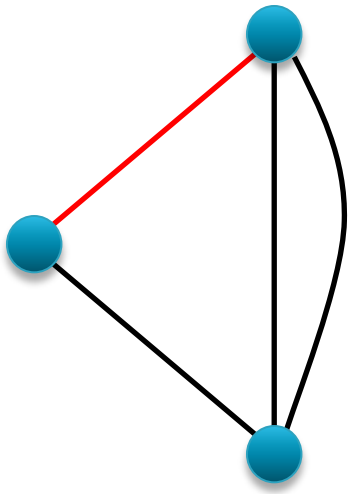
Min-Cut Randomized Algorithm



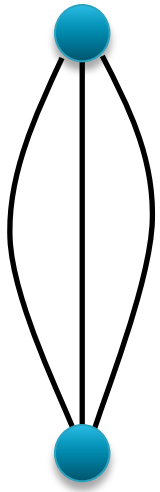
Min-Cut Randomized Algorithm



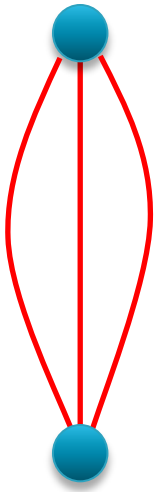
Min-Cut Randomized Algorithm



Min-Cut Randomized Algorithm



Min-Cut Randomized Algorithm



Weight = 3

Analysis of the Algorithm

Assume that the graph has a min-cut set of k edges.
We compute the probability of finding one such set C.

Lemma

If the edge contracted does not belong to C, no other edge eliminated in that step belongs to C.

Proof.

A contraction eliminates a set of parallel edges (edges connecting one pair of vertices).
Parallel edges either all belong, or don't belong to C. □

Min-Cut Randomized Algorithm

- **Lemma**: Vertex contraction does not reduce the size of the minimum cut-set

- **Proof**: Every cut-set in the new graph was a cut-set in the original graph

Let $E_i =$ "the edge contracted in iteration i is not in C."
Let $F_i = \cap_{j=1}^{i} E_j =$ "no edge of C was contracted in the first i iterations".
We need to compute $Pr(F_{n-2})$

Since the minimum cut-set has k edges, all vertices have degree $\geq k$, and the graph has $\geq nk/2$ edges.
There are at least $nk/2$ edges in the graph, k edges are in C.
$Pr(E_1) = Pr(F_1) \geq 1 - \frac{2k}{nk} = 1 - \frac{2}{n}$.

Assume that the first contraction did not eliminate an edge of C (conditioning on the event $E_1 = F_1$).

After the first vertex contraction we are left with an $n-1$ node graph, with minimum cut set, and minimum degree $\geq k$.

The new graph has at least $k(n-1)/2$ edges.

$Pr(E_2 \mid F_1) \geq 1 - \frac{k}{k(n-1)/2} \geq 1 - \frac{2}{n-1}$.

Similarly,

$Pr(E_i \mid F_{i-1}) \geq 1 - \frac{k}{k(n-i+1)/2} = 1 - \frac{2}{n-i+1}$.

We need to compute

$$Pr(F_{n-2})$$

We use

$$Pr(A \cap B) = Pr(A \mid B)Pr(B)$$

$$Pr(F_{n-2}) =$$

$$Pr(E_{n-2} \cap F_{n-3}) = Pr(E_{n-2} \mid F_{n-3})Pr(F_{n-3}) =$$

$$Pr(E_{n-2} \mid F_{n-3})Pr(E_{n-3} \mid F_{n-4}) \ldots . Pr(E_2 \mid F_1)Pr(F_1) \geq$$

$$\geq \Pi_{i=1}^{n-2}(1 - \frac{2}{n-i+1}) = \Pi_{i=1}^{n-2}(\frac{n-i-1}{n-i+1})$$

$$= (\frac{n-2}{n})(\frac{n-3}{n-1})(\frac{n-4}{n-2}) \ldots (\frac{4}{6})(\frac{3}{5})(\frac{2}{4})(\frac{1}{3}) = \frac{2}{n(n-1)}.$$

BPP Class

- **B**ounded-error, **P**robabilistic, **P**olynomial time
  - the class of *decision problems* solvable by a *probabilistic Turing machine* in *polynomial time*, with an *error probability* of *at most 1/3* for all instances.

- The choice of 1/3 is *arbitrary*. If we chose any constant between 0 and 1/2 (exclusive), the set **BPP** would be *unchanged*; however, this constant must be *independent of the input*.

- **BPP** is one of the largest *practical* classes of problems.

BQP Class

- **B**ounded-error, **Q**uantum, **P**olynomial time
  - the class of *decision problems* solvable by a *Quantum computer* in *polynomial time*, with an *error probability* of *at most 1/3* for all instances.

- A computation on a quantum computer ends with a *measurement*, which leads to a *collapse* of the quantum state to one of the basis states.
  - the quantum state is *measured* to be in the *correct state* with *high probability*.

- Some problems of practical interest are known to be in **BQP** but suspected to be outside **P**
  - Integer factorization
  - Discrete logarithm
  - Simulation of quantum systems

Complexity Classes