Randomized Algorithms

- Algorithms that make random choices during the computation

- Often faster, simpler than traditional algorithms

Miller-Rabin primality test

Input: n-bit number x.

Goal: decide whether x is a prime number or not.

- Extremely important problem: many applications in cryptography.
- There is a deterministic polynomial time algorithm (AKS-2000), running time is $O(n^{12})$

The test (running time $O(n^2)$):

1. Express $x - 1 = 2^s \cdot d$, where d is odd.

2. Pick $a \in \{1,2,\ldots,x-1\}$ uniformly at random.

3. If for some $t = 1,2,\ldots,s$, $a^{2^t \cdot d} = 1 \mod x$, yet $a^{2^{t-1} \cdot d} \neq -1 \mod x$, conclude that x is not prime. Otherwise conclude that x is prime.

Theorem: If x is prime, the test concludes that x is prime with probability 1. If x is not prime, the test concludes not prime with probability at least 3/4.

Min-Cut

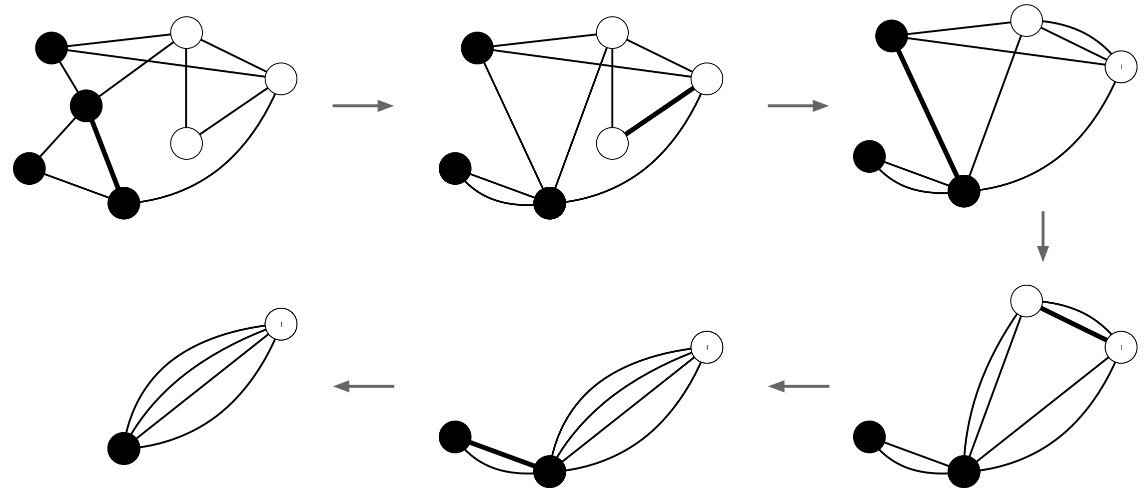
Input: An undirected graph.

Goal: Partition the vertices of the graph in two sets A, B, to minimize the number of edges going from A to B.

- You can use flows and cuts, but there is a simpler randomized algorithm

Karger's Algorithm:

1. In each step, pick a uniformly random edge and contract it.
2. Stop when you have just two vertices.
3. Output the corresponding cut.

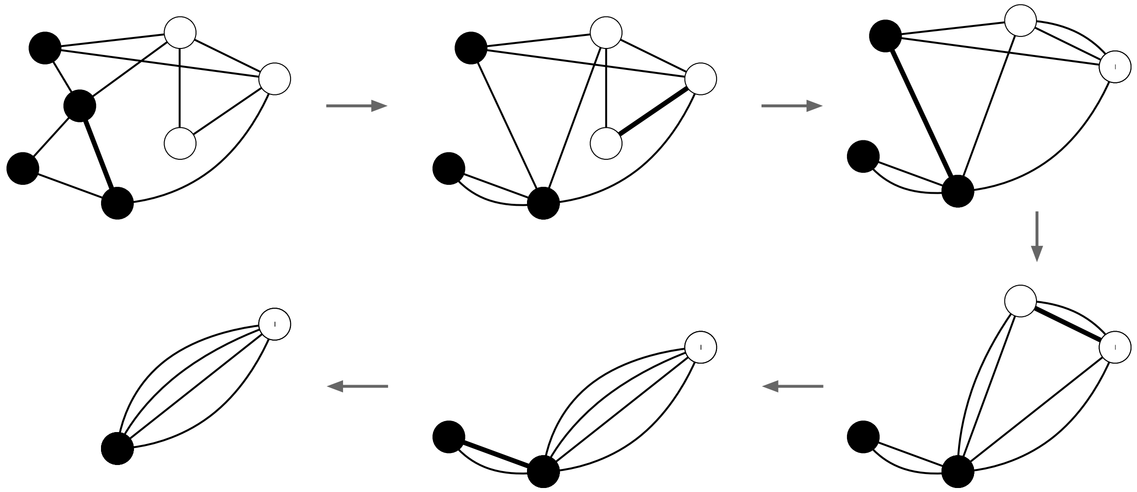


Karger's Algorithm:

1. In each step, pick a uniformly random edge and contract it.

2. Stop when you have just two vertices.

3. Output the corresponding cut.



Thm: The algorithm finds the min-cut with probability at least $2/(n(n-1))$.

Pf:

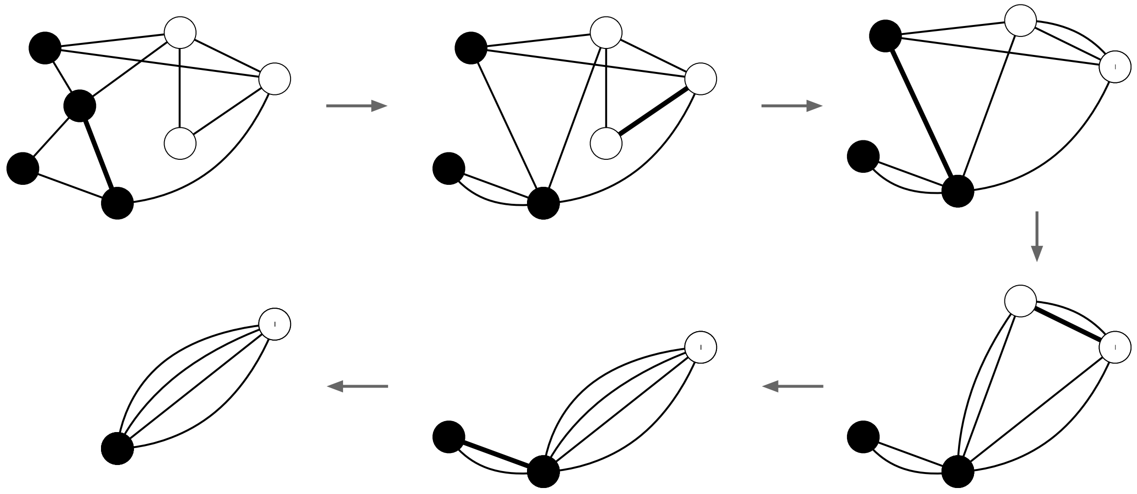
- Suppose the min-cut cuts k edges.
- Then every vertex must degree $\geq k$, or else that vertex would already give a smaller min-cut.
- So, the number of edges in the graph is at least $nk/2$.
- The probability we pick one of the edges of the min-cut is at most $k/(nk/2) = 2/n$.
- The probability that an edge of the min-cut is never picked is at least $(1-2/n)(1-2/(n-1))\ldots(1-2/3)$ $= ((n-2)/n) \cdot ((n-3)/(n-1)) \cdot ((n-4)/(n-2))\ldots \;= 2/(n(n-1))$.

Karger's Algorithm:

1. In each step, pick a uniformly random edge and contract it.

2. Stop when you have just two vertices.

3. Output the corresponding cut.



Final algorithm: Repeat the above algorithm $100n(n-1)$ times. Output the best cut that you find.

Graph coloring

Input: An undirected graph.
Goal: Find a 3-coloring of vertices that maximizes the number of edges that get 2 colors.

Algorithm:
Randomly color the vertices of the graph red,blue,green.

Thm: The expected number of vertices that are properly colored is at least $2m/3$.
Pf: For each edge e, define $X_e = 1$ if the edge e gets two colors, and $X_e = 0$ otherwise.
$\mathbb{E}[X_e] = \Pr[X_e = 1] \cdot 1 = 2/3$.
So, by linearity of expectation,
$\mathbb{E}[\sum_e X_e] = \sum_e \mathbb{E}[X_e] = 2m/3$.

No known poly time algorithm achieves $> 2m/3$**.**

Dominating set

Input: An undirected graph, every vertex has degree $\geq \Delta$.
Goal: Find a small set of vertices S such that every vertex is either in S or is a neighbor of S.

Algorithm:

1. Randomly include each vertex in the set X, with probability p.
2. Let Y be the set vertices not in X and not a neighbor of X.
3. Output $X \cup Y$.

Claim: The expected size of $X \cup Y$ is at most $pn + n(1-p)^{1+\Delta} \leq pn + e^{-p(1+\Delta)}n$.
Set $p = \ln(1+\Delta)/(1+\Delta)$, to get expected size at most $n(1+\ln(1+\Delta))/(1+\Delta)$.

Pf of Claim:

1. The expected size of X is pn.
2. For each vertex, the probability that it is included in Y is at most $(1-p)^{1+\Delta}$.
3. So the expected size of Y is $n(1-p)^{1+\Delta}$.

Matrix product checking in $O(n^2)$ time.

Input: $n \times n$ matrices A, B, C
Goal: Check that $AB = C$

Algorithm:
1. Pick $x \in \{0,1\}^n$ uniformly at random.
2. Check $ABx = Cx$

Claim: If $AB \neq C$, then $\Pr[ABx = Cx] \leq 1/2$.

Pf of Claim:
Let $D = (AB - C)$
Suppose $D_{i,j} \neq 0$, then $(Dx)_i = \sum_k D_{i,k}x_k = D_{i,j}x_j + \sum_{k \neq j} D_{i,k}x_k$, so for every fixing of $\sum_{k \neq j} D_{i,k}x_k$, the probability that $(Dx)_i = 0$ is at most $1/2$.