Randomized algorithms

Randomness in algorithms

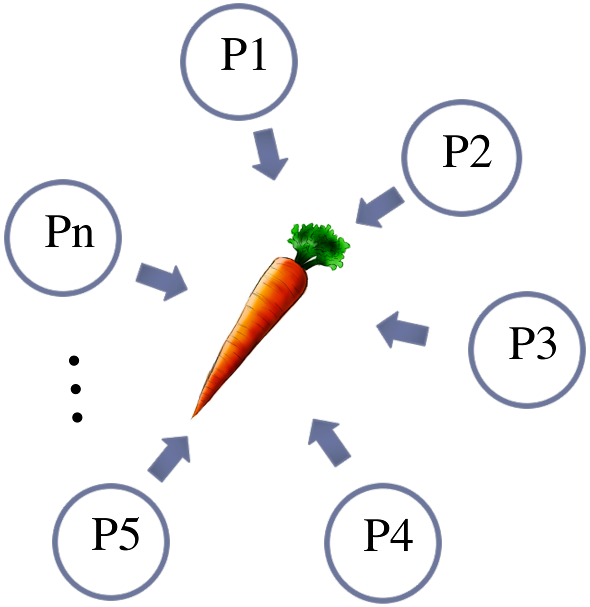
- Data is random $\Rightarrow$ average-case analysis of algorithms
- Algorithm is non-deterministic, uses randomness $\Rightarrow$ **randomized algorithms**

Some applications of randomization

- Universal hashing: helps to avoid dependency on data distribution, insures security (e.g. adversary-made worst cases)
- Locality-sensitive hashing: helps "sketching", i.e. representing big objects by small objects without much distortion of distances
- Online algorithms: improves the competitiveness
- Approximate counting and other streaming algorithms: drastically reduces spaces

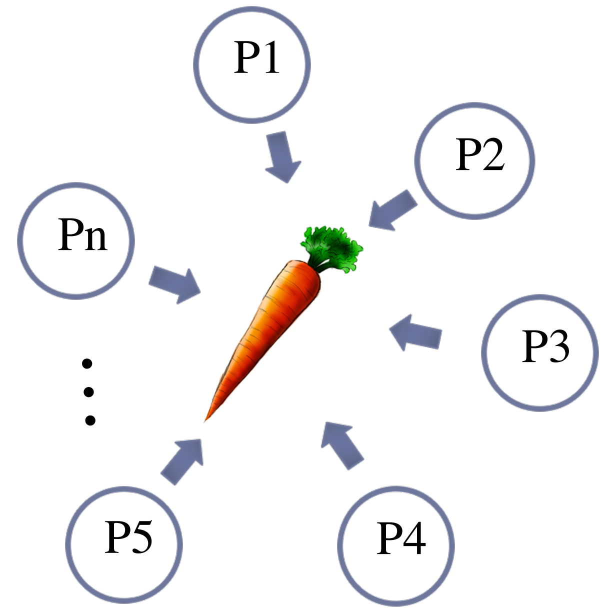
Accessing an exclusive resource by non-communicating processes

- *n* non-communicating processes want to access a resource in rounds
- if two processes ask for access at the same round, none of them gets access
- *What would be the good strategy?*



Accessing an exclusive resource by non-communicating processes

- *n* non-communicating processes want to access a resource in rounds
- if two processes ask for access at the same round, none of them gets access
- *What would be the good strategy?*
- Each process ask access with probability p ("*symmetry breaking*")
- optimum: $p = 1/n$
- then each process will get access on average between $1/en$ and $1/2n$ of time (depending on n)
- with probability at least $(1 - 1/n)$ all processes access R at least once in $O(n \log n)$ rounds (about $2en \cdot \ln(n)$)



Besides …

- … randomized algorithms are usually *simple* (to implement, not to understand) and, as a consequence, *efficient*

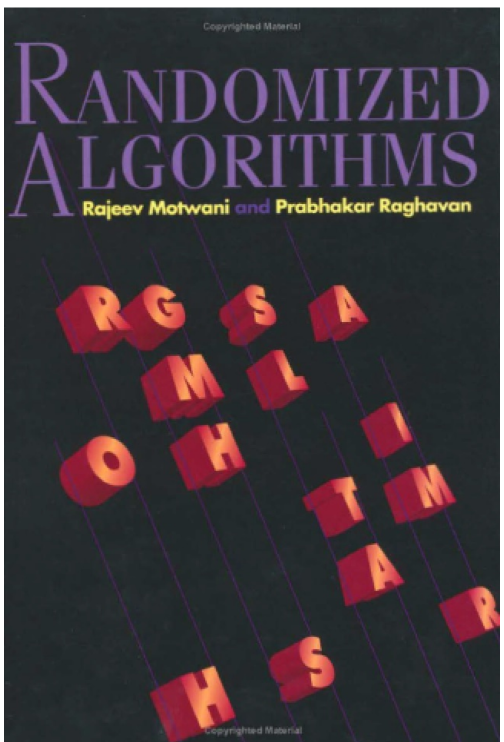
Two types of randomized algorithms

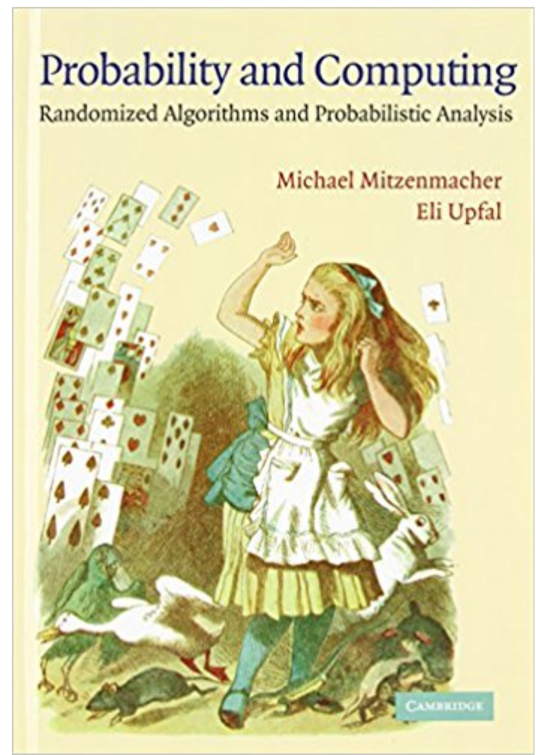
- Las Vegas: always correct (but fast *in expectation*, e.g. *Randomized QUICKSORT*)

- Monte Carlo: admits a small error probability

Books on randomized algorithms

- Motwani, Raghavan, Randomized algorithms, Cambridge University Press, 1995
- Mitzenmaher, Upfal, Probability and Computing: Randomzied Algorithms and Probabilistic Analysis, Cambridge University Press, 2005





Monte Carlo: simple example

- Given a (very large) array of numbers, return some array element larger than the median

Edge connectivity (global min-cut)

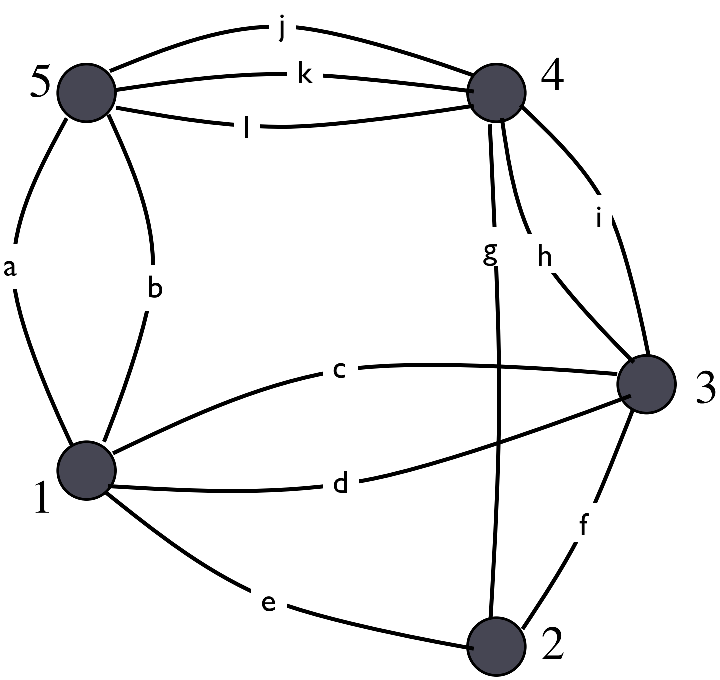
- Given an undirected graph $G = (V, E)$, a *cut* is $V = A \cup B$ $(A \cap B = \emptyset)$. What the minimal number of edges connecting the two parts of a cut of G?
- Can be solved in polynomial time:
  - $O(n^2 m)$ using Edmons-Karp
  - $O(n(n+m)\log n)$ using Stoer-Wagner
- A much simpler *randomized* algorithm exists due to David Karger (1992)

High-level idea

- Stoer-Wagner takes time $O((n+m)\log n)$ to find the pair of nodes to merge
- *Idea*: pick a random edge and merge endpoints (contract the edge)

Contraction algorithm

- Like Stoer-Wagner, the algorithm works with *multigraphs*. (The input graph G can be a multi-graph as well)

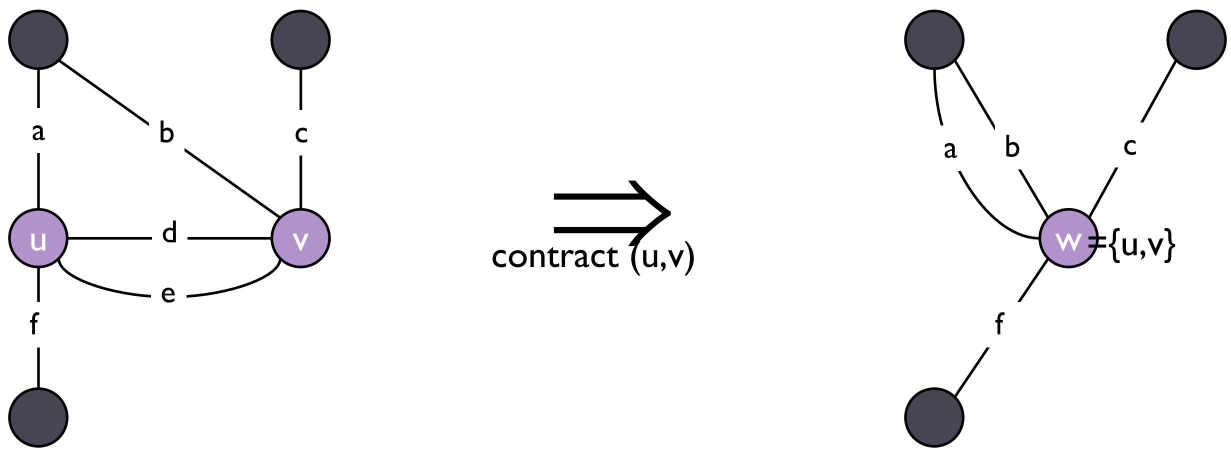


C={c,d,e,j,k,l}=({1,5},{2,3,4})

C={e,f,g}=({2},{1,3,4,5})

Contraction

- Contraction of an edge (u, v):
  - merge u and v into a single new node w
  - for all edges (t, u) or (t, v), update to (t, w)
  - keep parallel edges, but delete self-loops



- *Remark (cf Stoer-Wagner)*: Contraction can not decrease the min-cut value (but can increase)

Contraction algorithm

do

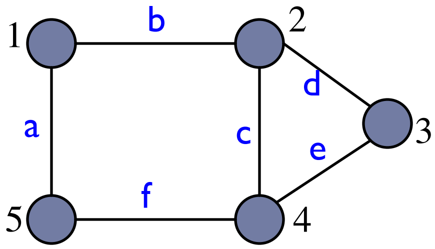
pick an edge $e = (u, v)$ uniformly at random;

contract (u, v)

until G has just two nodes s, t

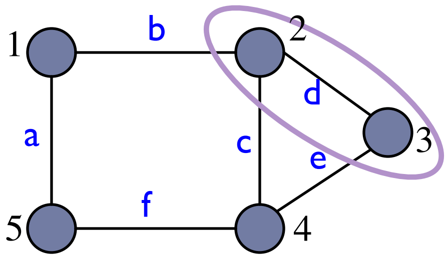
return the corresponding cut (nodes contracted to s and t respectively) and the number of crossing edges

Example



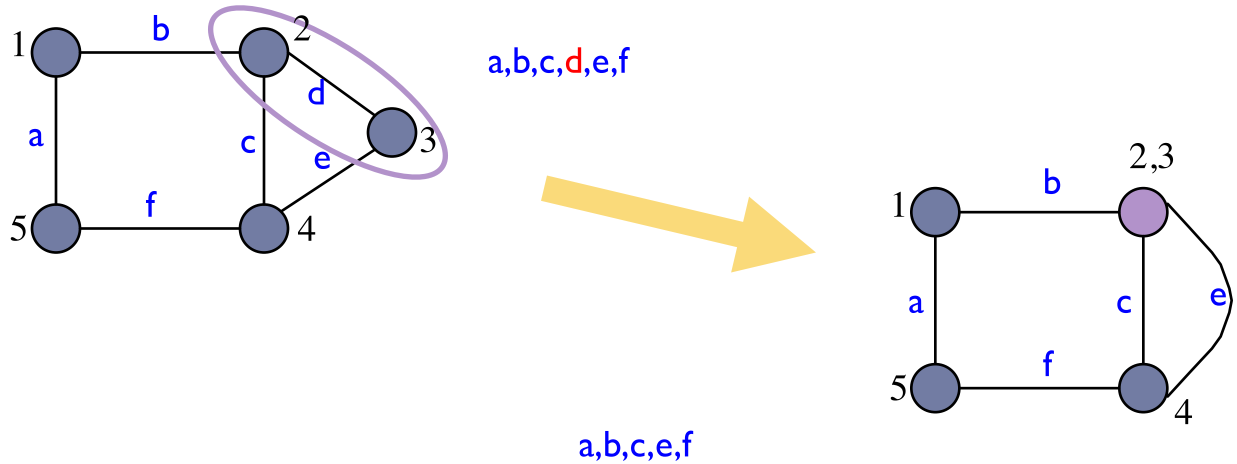
a,b,c,d,e,f

Example

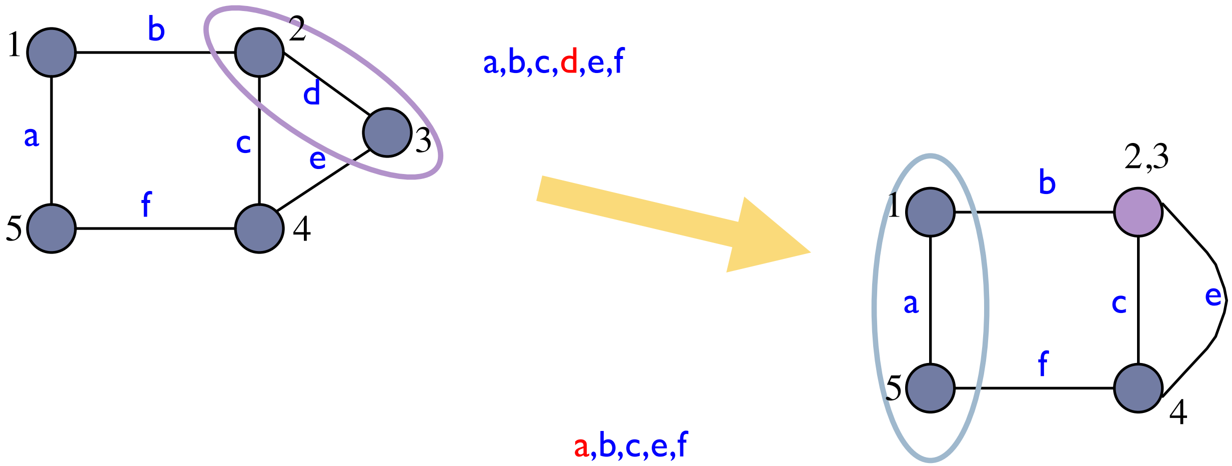


a,b,c,d,e,f

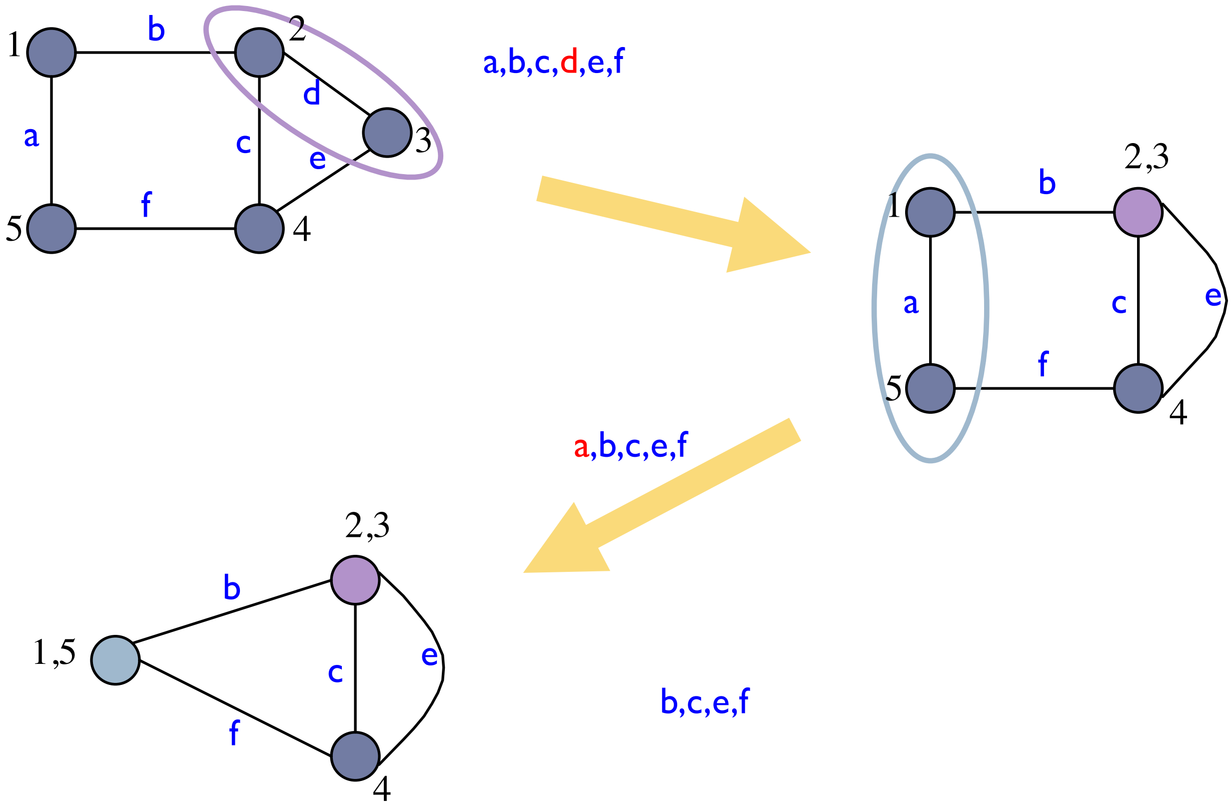
Example



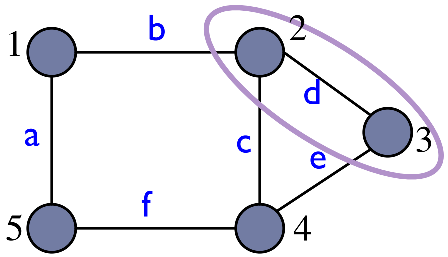
Example



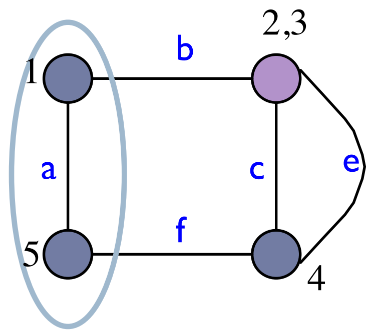
Example



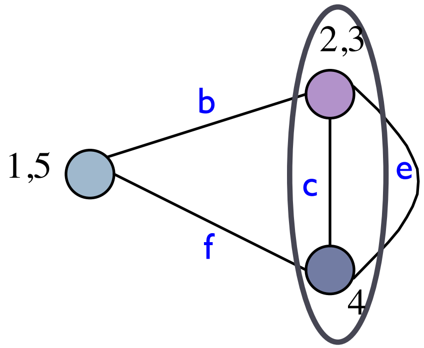
Example



a,b,c,d,e,f

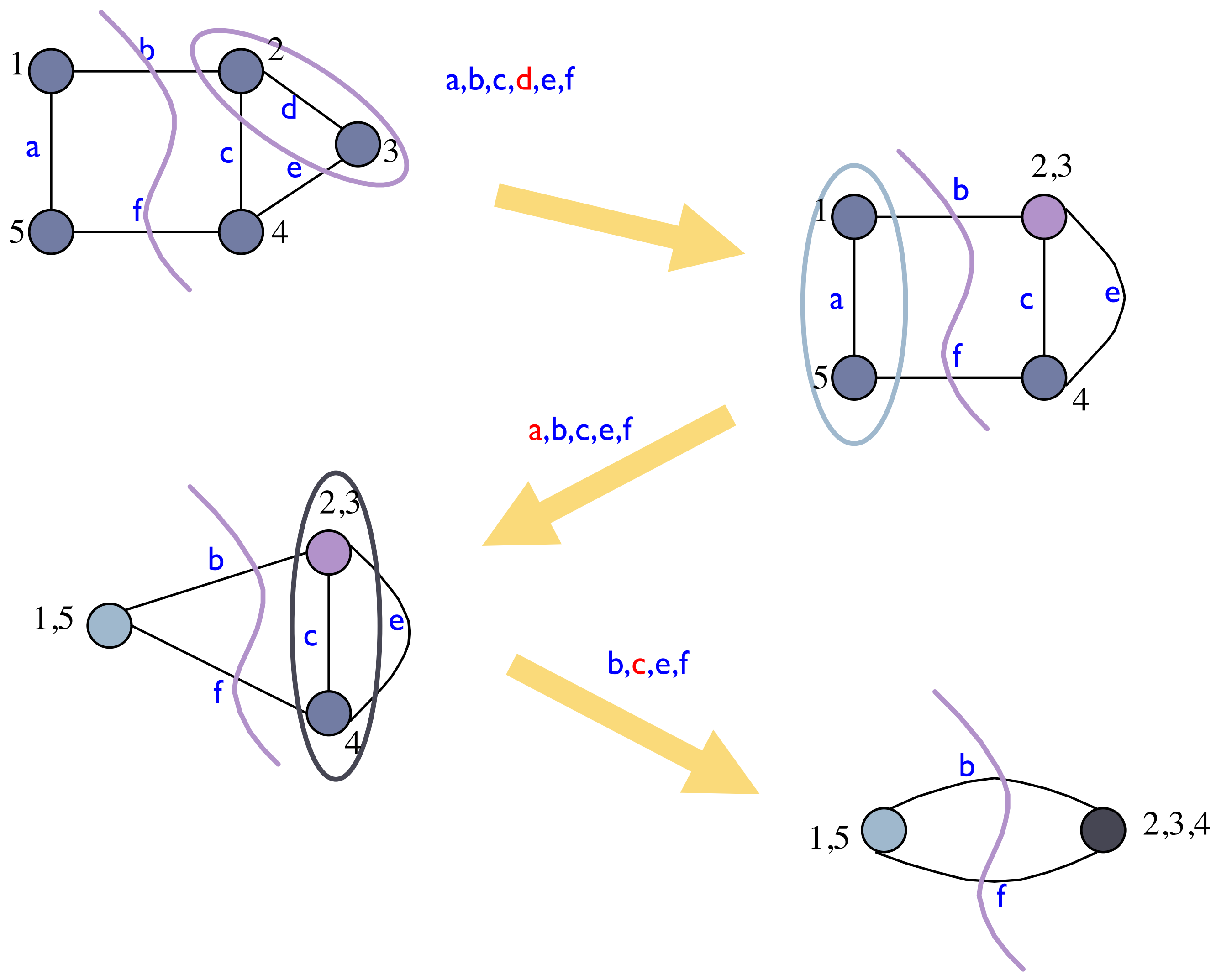


a,b,c,e,f



b,c,e,f

Example

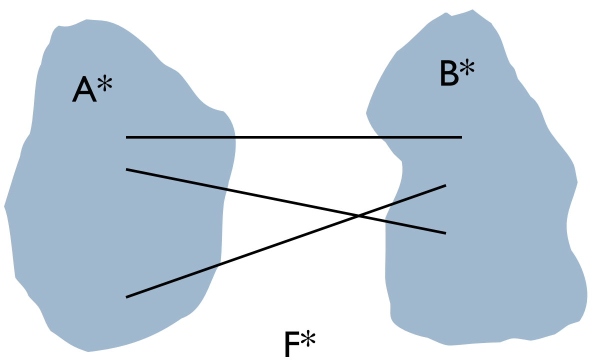


Analysis of the algorithm

Theorem: The contraction algorithm returns a min cut with probability $\geq \frac{2}{n(n-1)}$

Proof: Consider a global min-cut (A^*, B^*) of G. Let F^* be edges with one endpoint in A^* and the other in B^*. Let $k = F^* =$ size of min cut.

- In first step, algorithm contracts an edge in F^* probability $k/|E|$.
- Every node has degree $\geq k$ since otherwise (A^*, B^*) would not be min-cut $\Rightarrow |E| \geq \frac{kn}{2}$.
- Thus, algorithm contracts an edge in F^* with probability $\leq \frac{2}{n}$



Analysis of the algorithm (cont)

- Let G' be graph after j iterations. There are $n' = n - j$ supernodes.
- Suppose no edge in F^* has been contracted. The min-cut in G' is still k.
- Since value of min-cut is k, $|E'| \geq kn'/2$.
- Thus, algorithm contracts an edge in F^* with probability $\leq 2/n'$.
- Let E_j = event that an edge in F^* is not contracted in iteration j.

$$
\begin{aligned}
P[E_1 \cap E_2 \cap \cdots E_{n-2}] &= P[E_1] \cdot P[E_2|E_1] \cdot \cdots \cdot P[E_{n-2}|E_1 \cap E_2 \cap \cdots E_{n-3}] \\
&\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{4}\right)\left(1 - \frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \geq \frac{2}{n^2}
\end{aligned}
$$

Contraction algorithm: amplification

Amplification: To amplify the probability of success, run the contraction algorithm many times and take the best cut!

Claim: If we repeat the contraction algorithm $n^2 \ln n$ times with independent random choices, the probability of failing to find the global min-cut is at most $1/n^2$.

Proof: By independence, the probability of failure is at most

$$\left(1 - \frac{2}{n^2}\right)^{n^2 \ln n} = \left(\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}}\right)^{2 \ln n} \underset{(1-1/x)^x \le 1/e}{\le} (e^{-1})^{2 \ln n} = \frac{1}{n^2}$$

Contraction algorithm: complexity

- Contraction algorithm: $O(m \cdot n^2 \cdot \log(n)) = O(n^4 \cdot \log(n))$

- Solution by network flow requires time $O(n^4)$

- However, the contraction algorithm can be improved to $O(m \cdot \log^3(n))$ [Karger, Stein, Journal of the ACM, 1996; Karger, Journal of the ACM, 2000]