Advanced Algorithmic Techniques (COMP523)

Randomised Algorithms 2

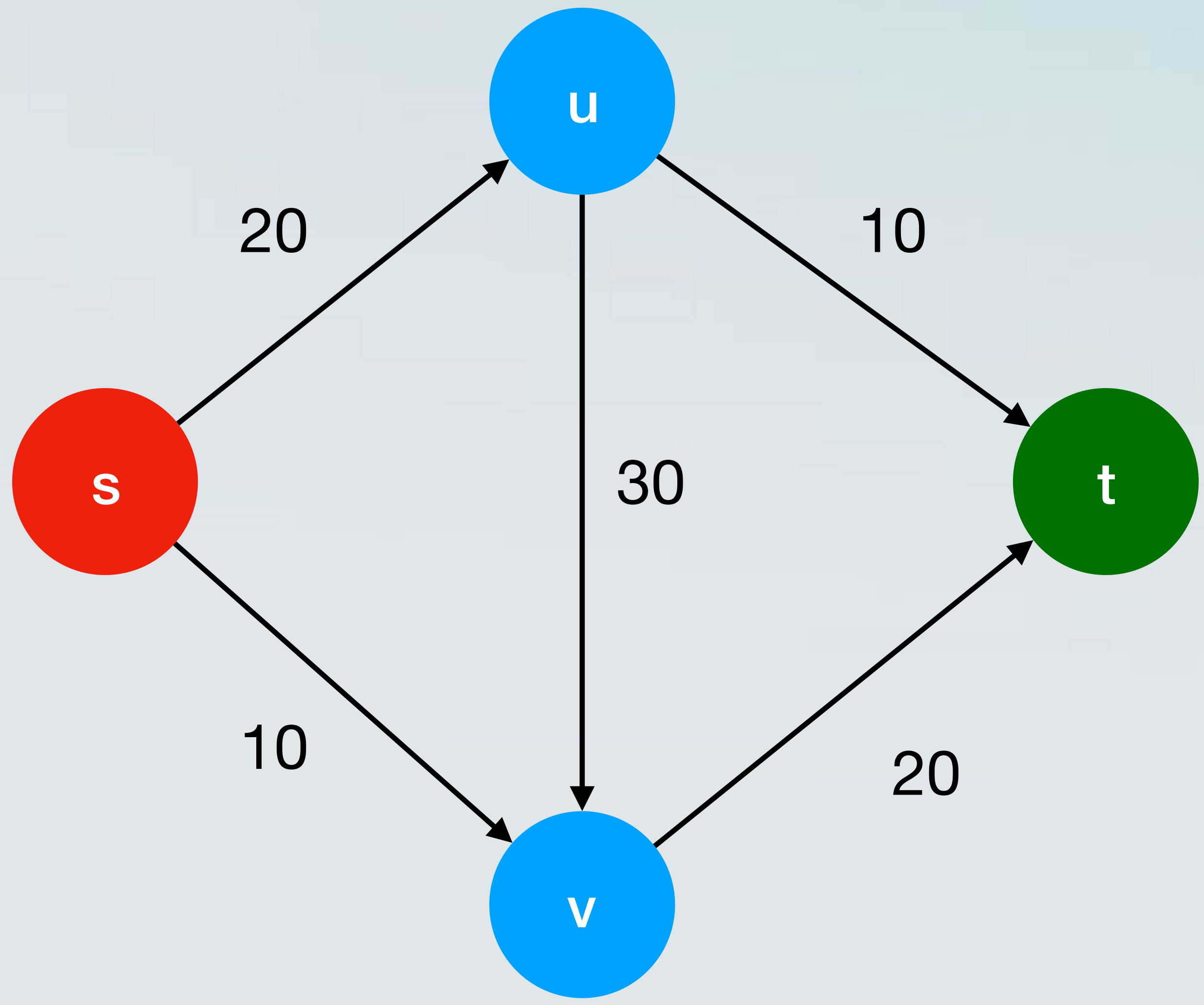
Recap and plan

- **Previous lecture:**
  - Probabilities background.
- **This lecture:**
  - Randomised global cuts in multi-graphs.

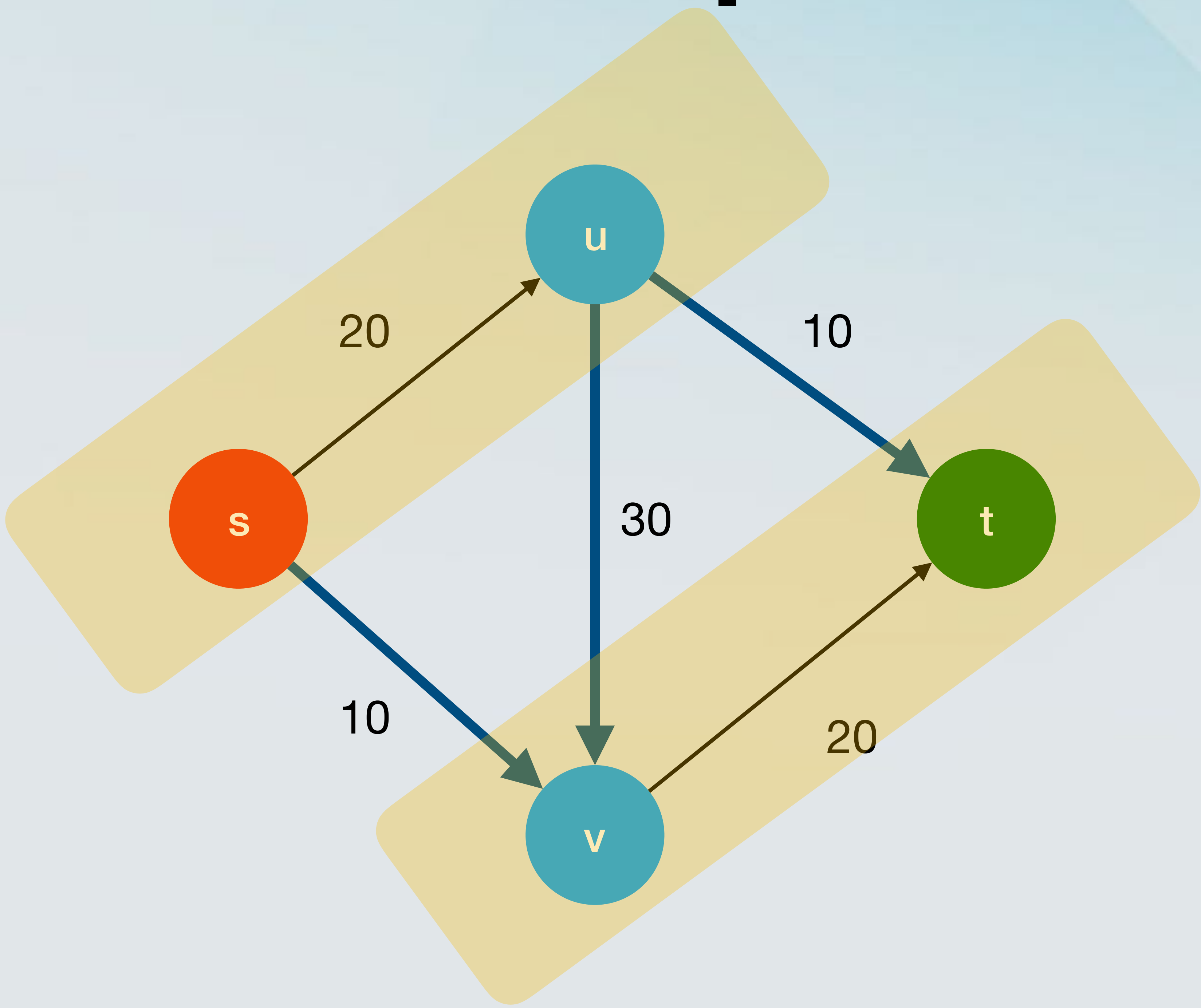
Minimum Cut

- A *cut* C is a partition of the nodes of G into two sets S and T, such that s is in S and t is in T.
- The capacity c(S,T) of a cut C is the sum of capacities of all edges "out of S"
  - these are edges (u, v) where u is in S and v is in T.

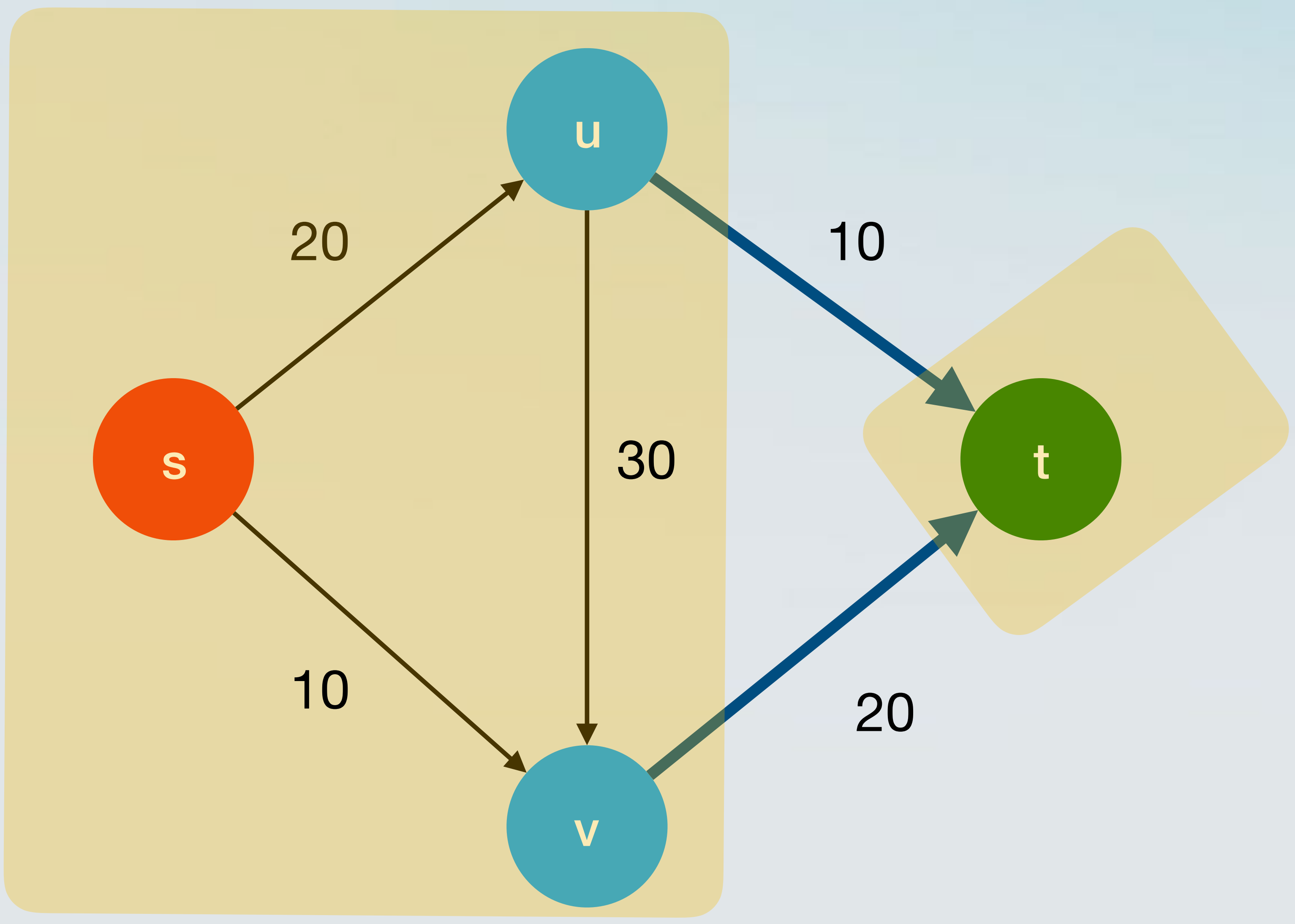
Example



Example



Example



Global Minimum Cut

Global Minimum Cut

- We are given an *undirected* graph G=(V, E).

Global Minimum Cut

- We are given an *undirected* graph $G=(V, E)$.
- A *cut* of G is a partition of the nodes of the graph into two sets, A and B.

Global Minimum Cut

- We are given an *undirected* graph G=(V, E).
- A *cut* of G is a partition of the nodes of the graph into two sets, A and B.
- The size of a cut (A, B) is *the number of edges* with one endpoint in A and one endpoint in B.

Global Minimum Cut

- We are given an *undirected* graph G=(V, E).
- A *cut* of G is a partition of the nodes of the graph into two sets, A and B.
- The size of a cut (A, B) is *the number of edges* with one endpoint in A and one endpoint in B.
- A *global minimum cut* is a cut of minimum size.

Solving GMC

Solving GMC

- Theorem: There is a polynomial-time algorithm for finding a global minimum cut in an undirected graph G.

Solving GMC

- Theorem: There is a polynomial-time algorithm for finding a global minimum cut in an undirected graph G.
  - Idea: Turn the graph G into a flow network, and find a minimum s-t cut.

Solving GMC

- Theorem: There is a polynomial-time algorithm for finding a global minimum cut in an undirected graph G.

  - Idea: Turn the graph G into a flow network, and find a minimum s-t cut.

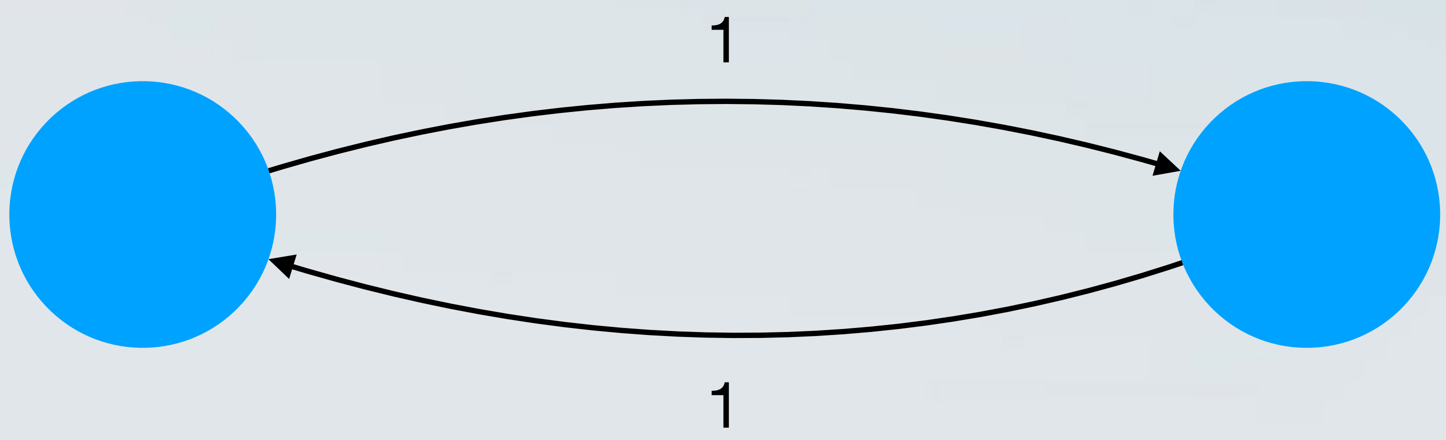
  - Replace every undirected edge with two directed edges, one in the forward and one in the backward direction. Set the capacity of those edges to be *1*.

Solving GMC

- Theorem: There is a polynomial-time algorithm for finding a global minimum cut in an undirected graph G.
  - Idea: Turn the graph G into a flow network, and find a minimum s-t cut.
  - Replace every undirected edge with two directed edges, one in the forward and one in the backward direction. Set the capacity of those edges to be *1*.
  - Pick two arbitrary nodes s, t in V, and find the minimum s-t cut (how?)

The procedure

The procedure



How many iterations of the max-flow algorithm?

How many iterations of the max-flow algorithm?

- We fix some s in V.

How many iterations of the max-flow algorithm?

- We fix some s in V.
- For every possible t in V (besides s), we run the algorithm.

How many iterations of the max-flow algorithm?

- We fix some s in V.
- For every possible t in V (besides s), we run the algorithm.
- In total, we will need *n-1* iterations.

How many iterations of the max-flow algorithm?

- We fix some s in V.
- For every possible t in V (besides s), we run the algorithm.
- In total, we will need *n-1* iterations.
- This is a polynomial-time algorithm, when the max-flow algorithm is polynomial-time.

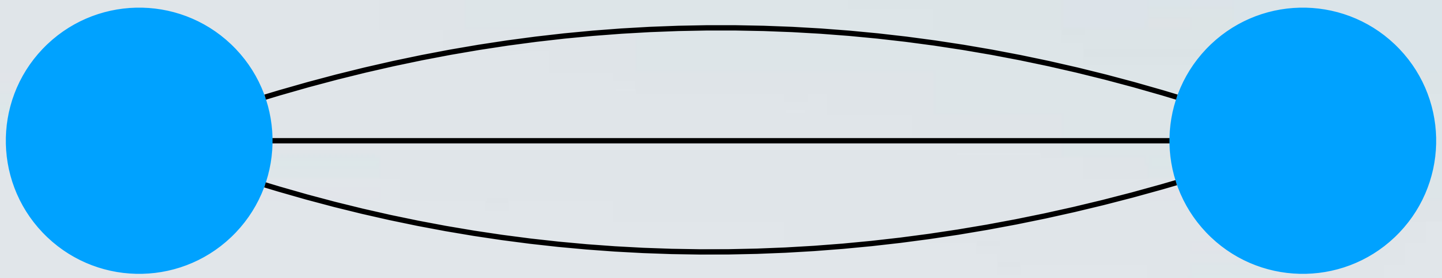
Global Minimum Cut

- We are given an *undirected* graph G=(V, E).
- A *cut* of G is a partition of the nodes of the graph into two sets, A and B.
- The size of a cut (A, B) is *the number of edges* with one endpoint in A and one endpoint in B.
- A *global minimum cut* is a cut of minimum size.

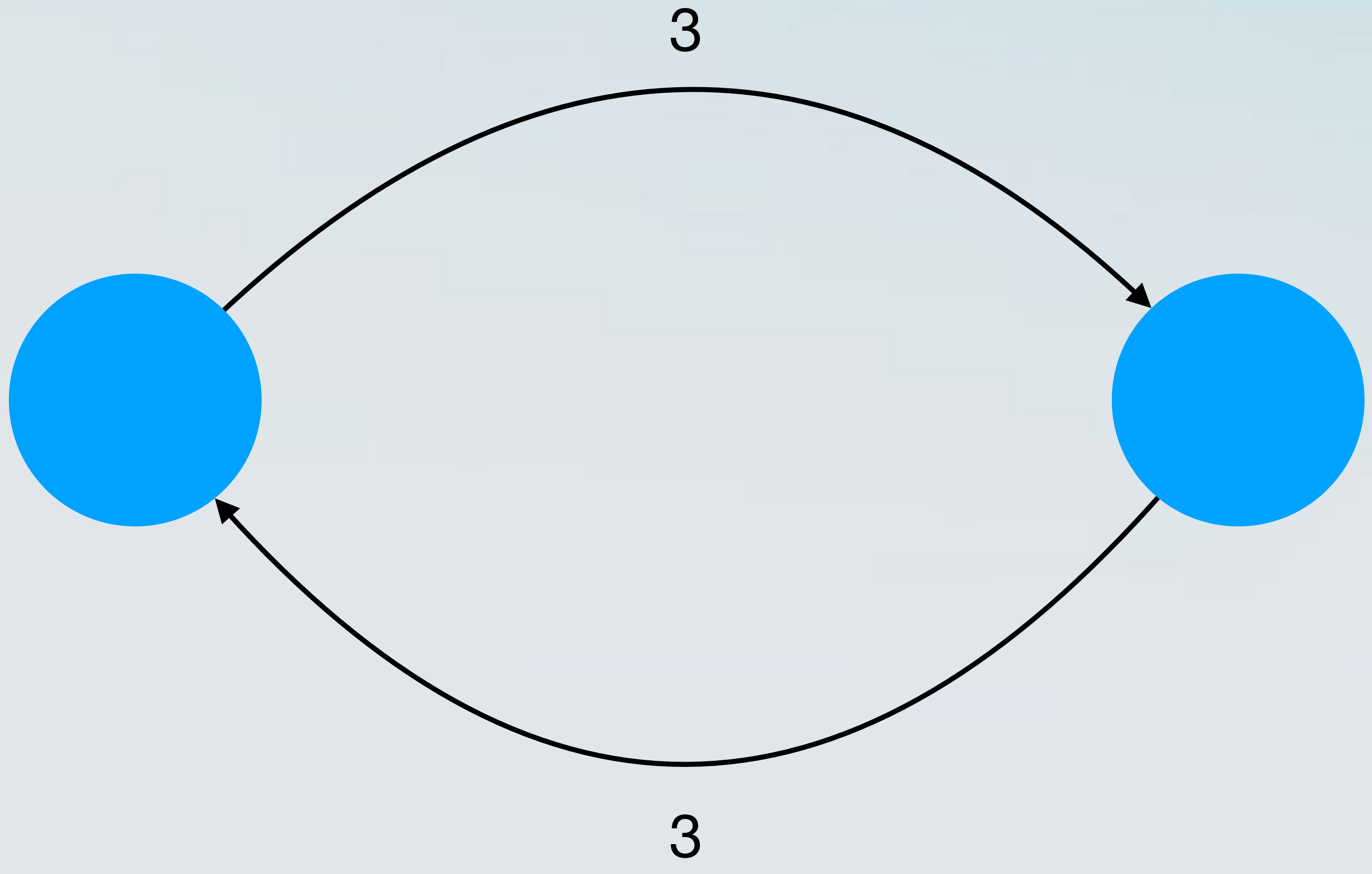
Global Minimum Cut

- We are given an *undirected* multigraph G=(V, E).
  - There can be multiple "parallel" edges between two nodes.
- A *cut* of G is a partition of the nodes of the graph into two sets, A and B.
- The size of a cut (A, B) is *the number of edges* with one endpoint in A and one endpoint in B.
- A *global minimum cut* is a cut of minimum size.

The procedure



The procedure



Is there a simpler solution?

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.
- It will produce the correct outcome *with high probability*.

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.
- It will produce the correct outcome *with high probability*.
  - Sometimes it might make a mistake!

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.
- It will produce the correct outcome *with high probability*.
  - Sometimes it might make a mistake!
  - But not too often.

The Contraction Algorithm

- Proposed in 1992 by David Karger.

The Contraction Algorithm

- Proposed in 1992 by David Karger.
- Idea:

The Contraction Algorithm

- Proposed in 1992 by David Karger.
- Idea:
  - Choose an edge of the graph *uniformly at random*.

The Contraction Algorithm

- Proposed in 1992 by David Karger.
- Idea:
  - Choose an edge of the graph *uniformly at random*.
  - Contract the edge.

The Contraction Algorithm

- Proposed in 1992 by David Karger.
- Idea:
  - Choose an edge of the graph *uniformly at random*.
  - Contract the edge.
    - Merge its endpoints (u, v) to a supernode w = {u, v}.

The Contraction Algorithm

- Proposed in 1992 by David Karger.
- Idea:
  - Choose an edge of the graph *uniformly at random*.
  - Contract the edge.
    - Merge its endpoints (u, v) to a supernode w = {u, v}.
    - Any edge (u, v) is removed.

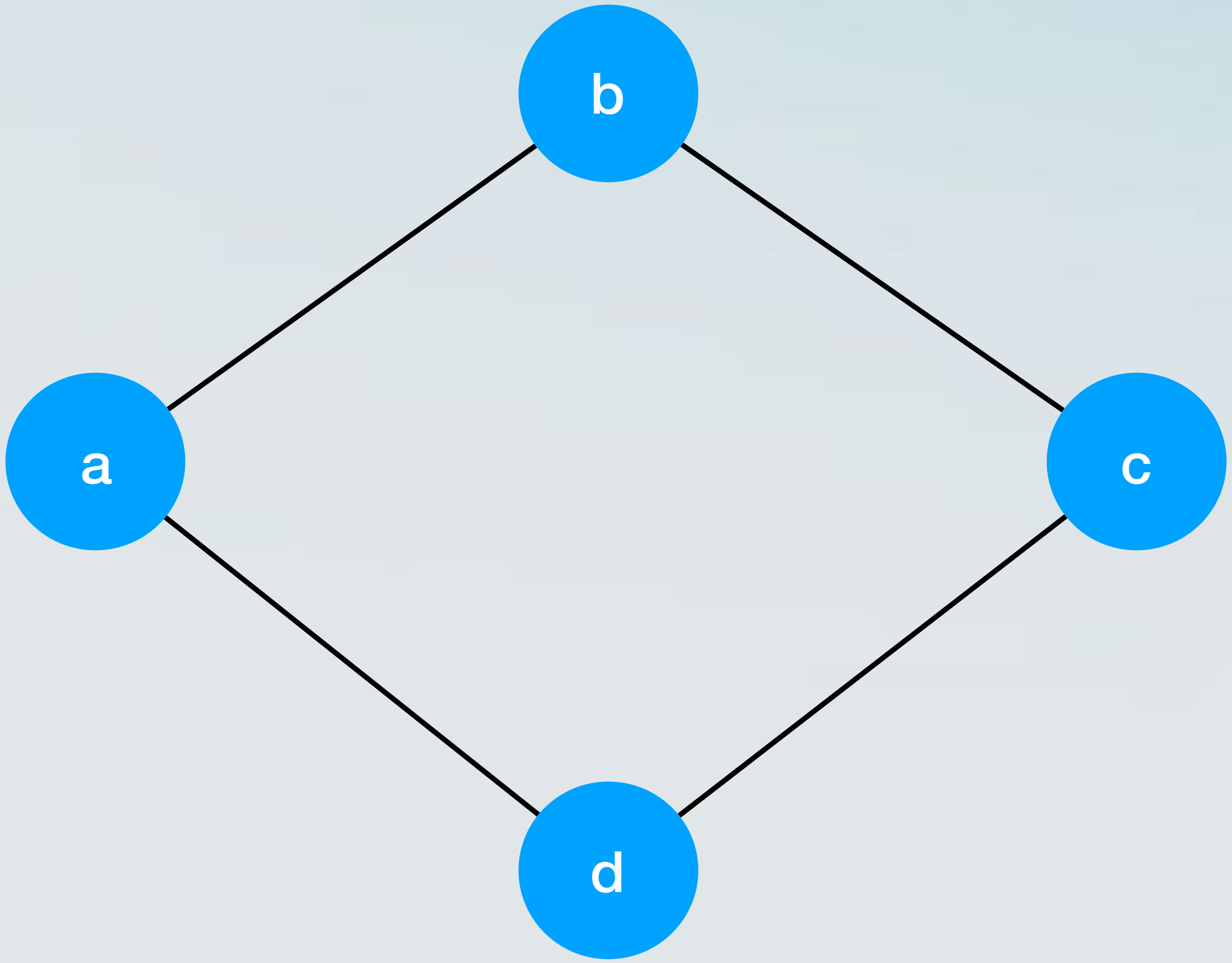
The Contraction Algorithm

- Proposed in 1992 by David Karger.
- Idea:
  - Choose an edge of the graph *uniformly at random*.
  - Contract the edge.
    - Merge its endpoints (u, v) to a supernode w = {u, v}.
    - Any edge (u, v) is removed.
    - Any edge (u, a) or (v, a) becomes (w, a).

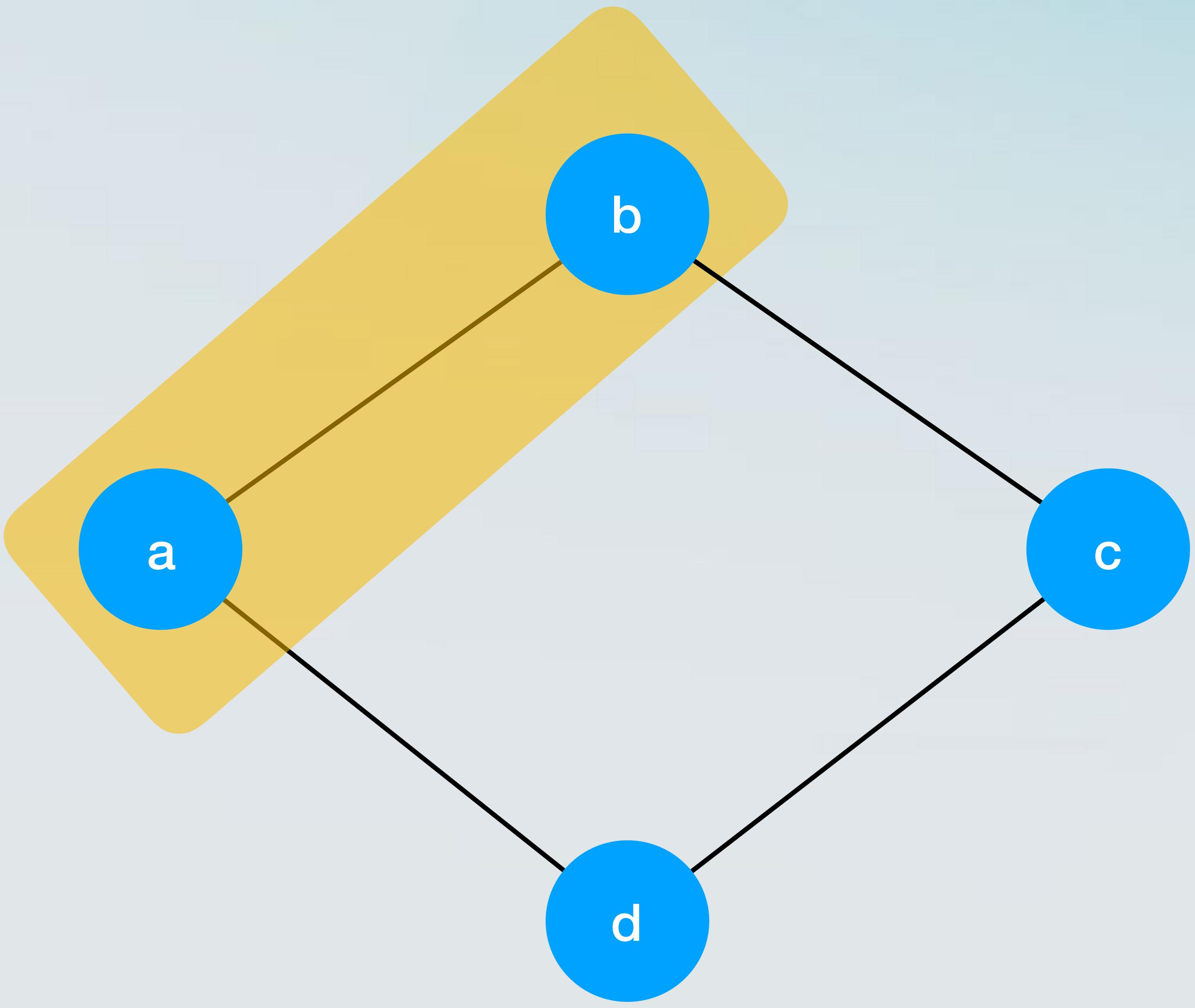
The Contraction Algorithm

- Proposed in 1992 by David Karger.
- Idea:
  - Choose an edge of the graph *uniformly at random*.
  - Contract the edge.
    - Merge its endpoints (u, v) to a supernode w = {u, v}.
    - Any edge (u, v) is removed.
    - Any edge (u, a) or (v, a) becomes (w, a).
  - When we are left with two supernodes w_1 and w_2, the corresponding sets of nodes are A and B.

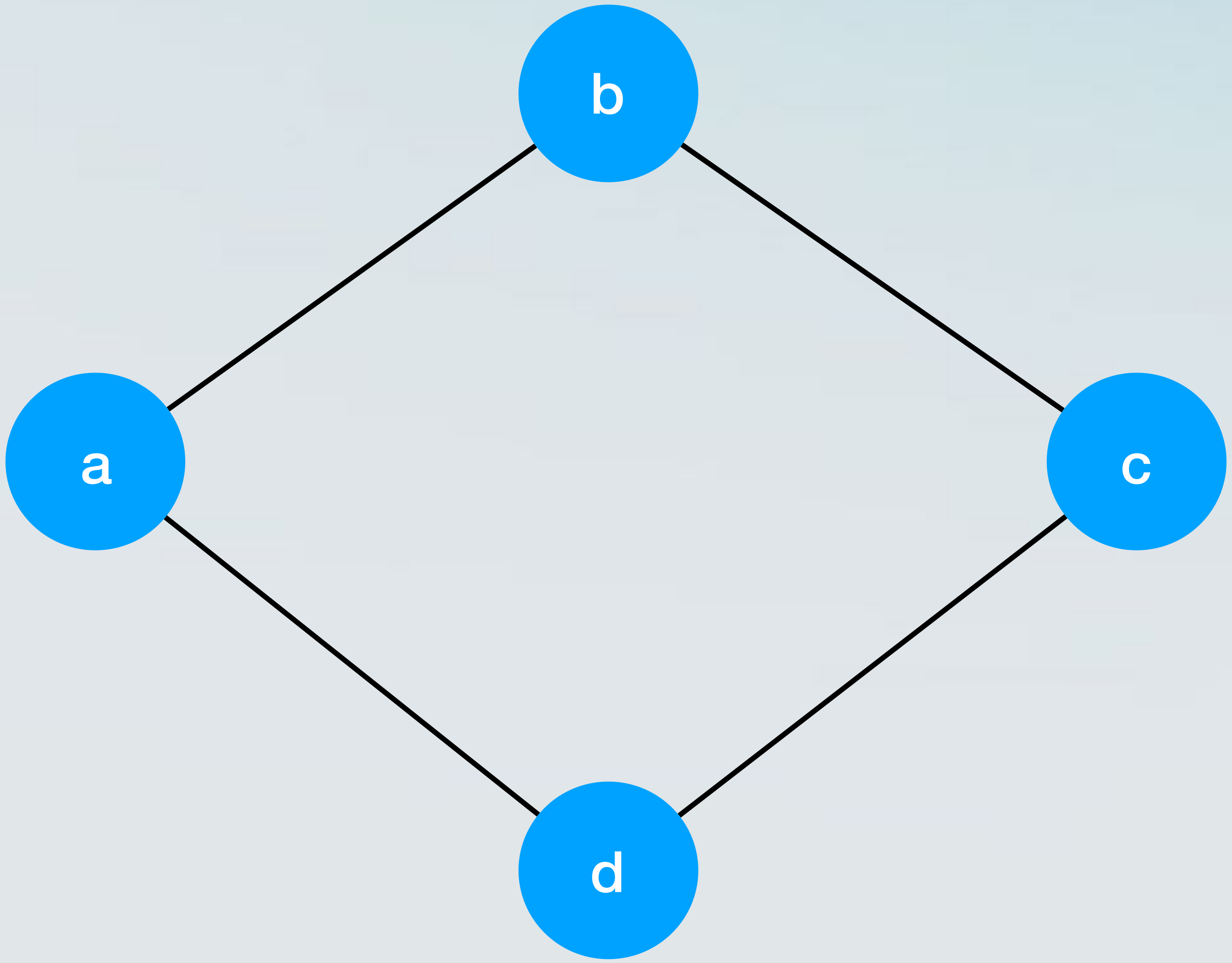
Example



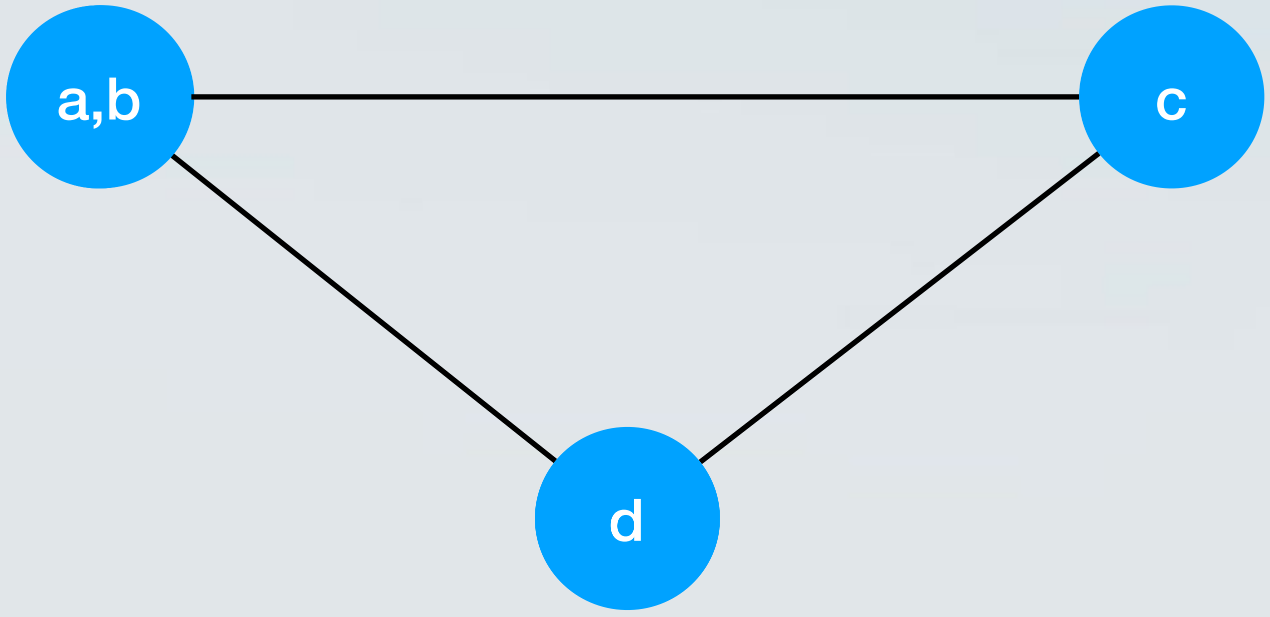
Example



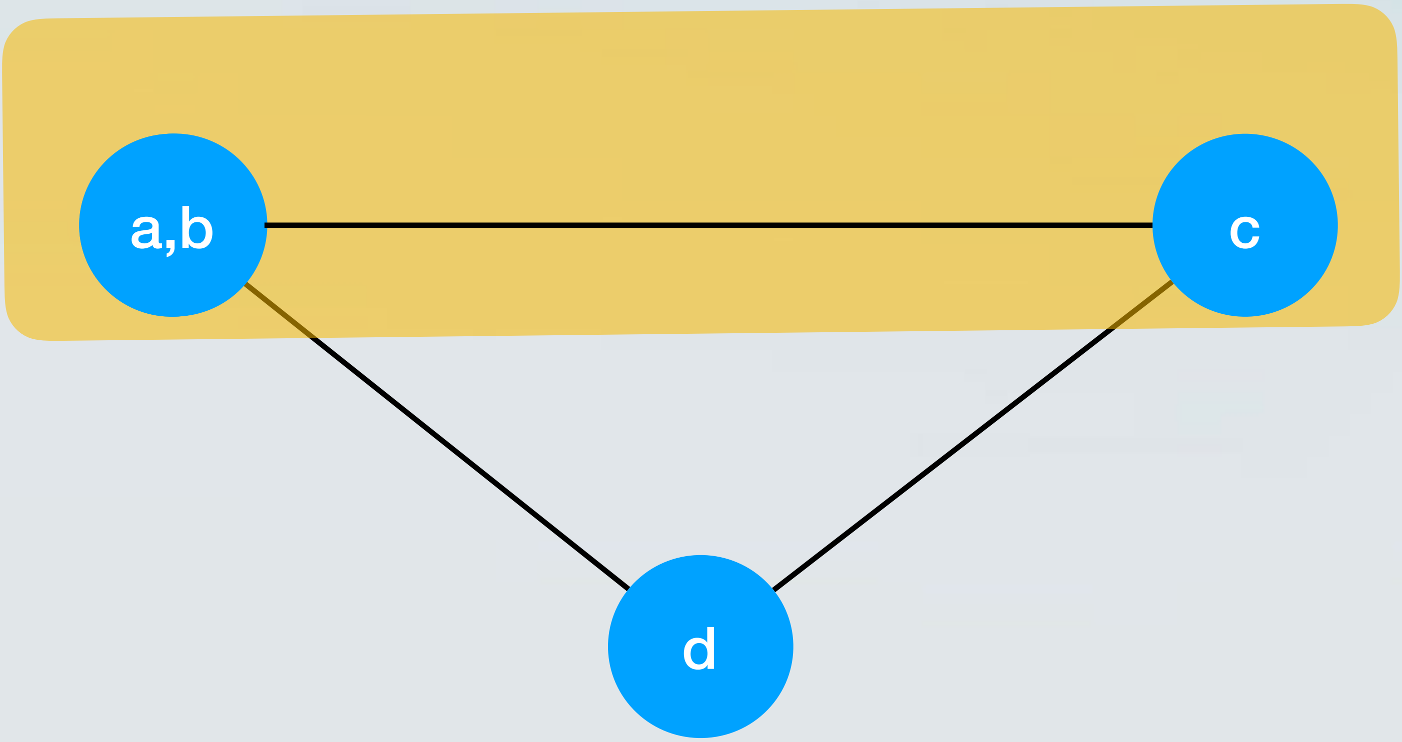
Example



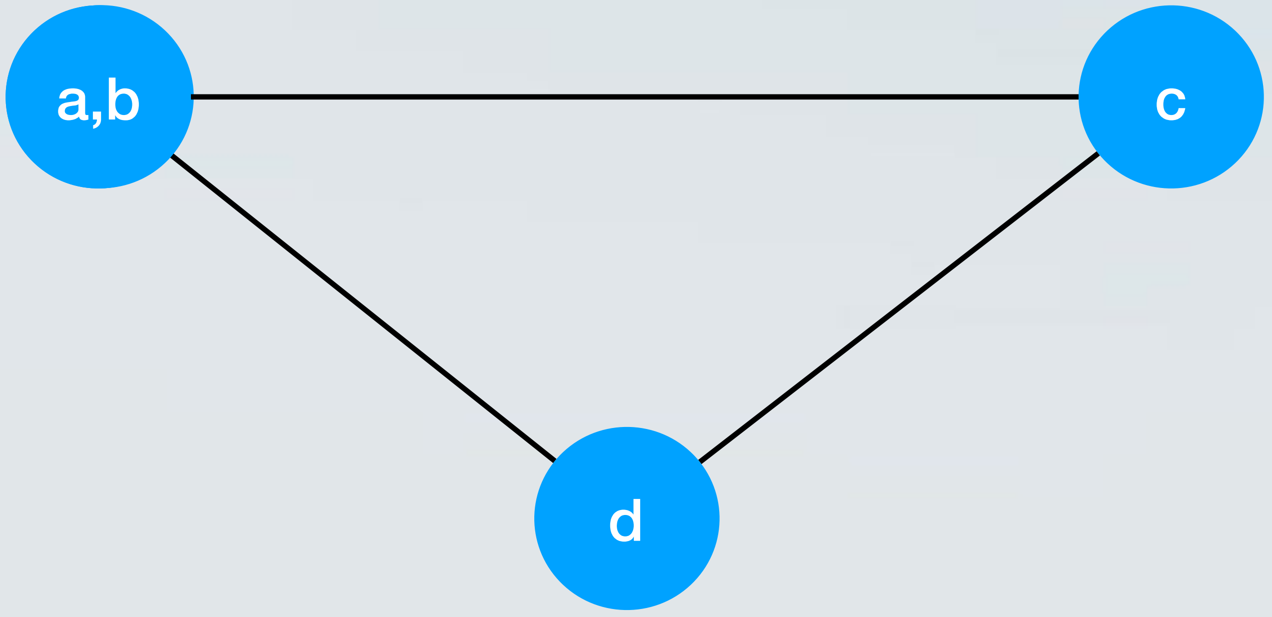
Example



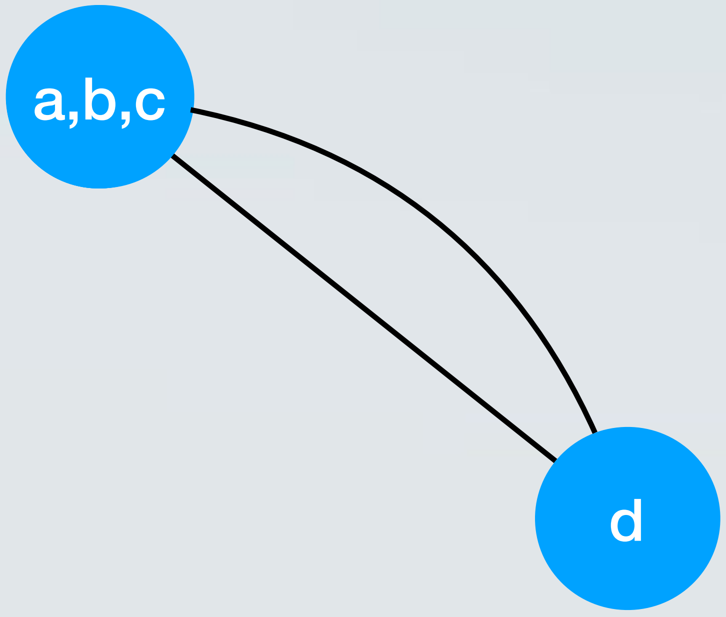
Example



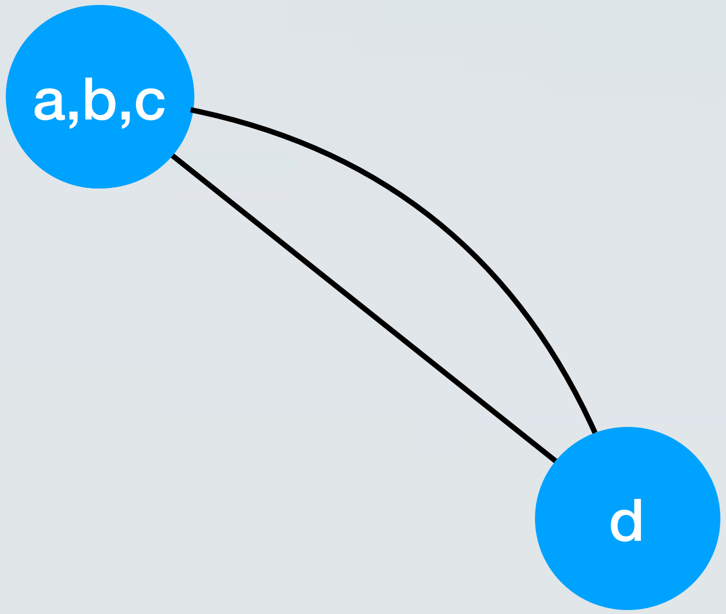
Example



Example



Example



A = {a, b, c}
B = {d}

The Contraction Algorithm

Contraction(G)

For each node v, record
    the set S(v) of nodes that have been contracted into v.
  Initially, S(v) = {v} for each v. */* no contractions so far */*

If G has two nodes v_1 and v_2, then return the cut {S(v_1), S(v_1)}.

Else, choose an edge e = (u, v) of G *uniformly at random*.
   Let G' be the graph resulting from contracting e, with a
   new node z_{uv} replacing u and v.

   Define S(z_{uv}) = S(u) ∪ S(v)
   Contraction(G')

EndIf

The analysis of the algorithm

The analysis of the algorithm

- Consider a global minimum (A, B) cut of G, and suppose that it has size k.

The analysis of the algorithm

- Consider a global minimum (A, B) cut of G, and suppose that it has size k.
  - In other words, there is a set F of edges with one endpoint in A and one endpoint in B, such that |F|=k.

The analysis of the algorithm

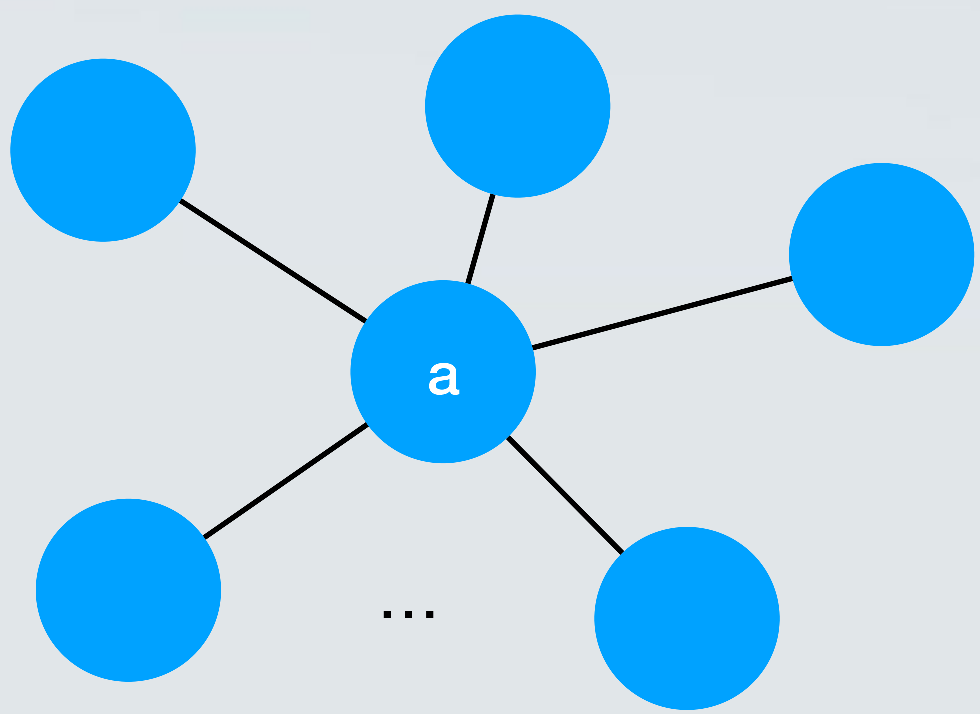
- Consider a global minimum (A, B) cut of G, and suppose that it has size k.
  - In other words, there is a set F of edges with one endpoint in A and one endpoint in B, such that |F|=k.
  - We will prove that the contraction algorithm outputs the cut (A, B) with *high probability*.

A first observation

- The *maximum degree* in G is at least k.
  - (Why?)

A first observation

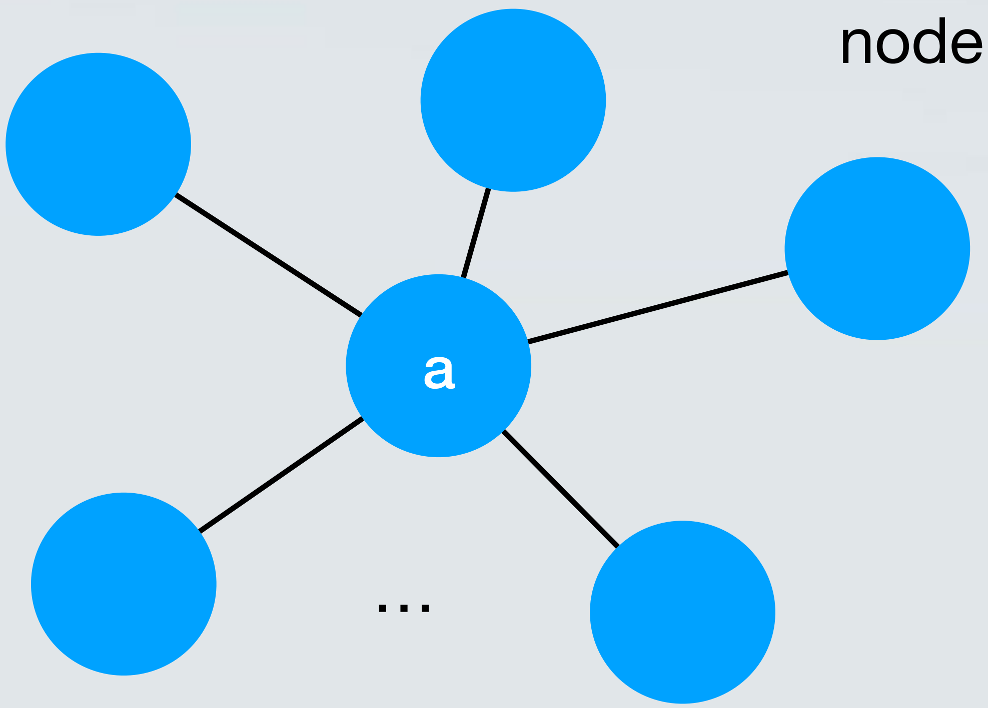
- The *maximum degree* in G is at least k.
  - (Why?)



A first observation

- The *maximum degree* in G is at least k.
  - (Why?)

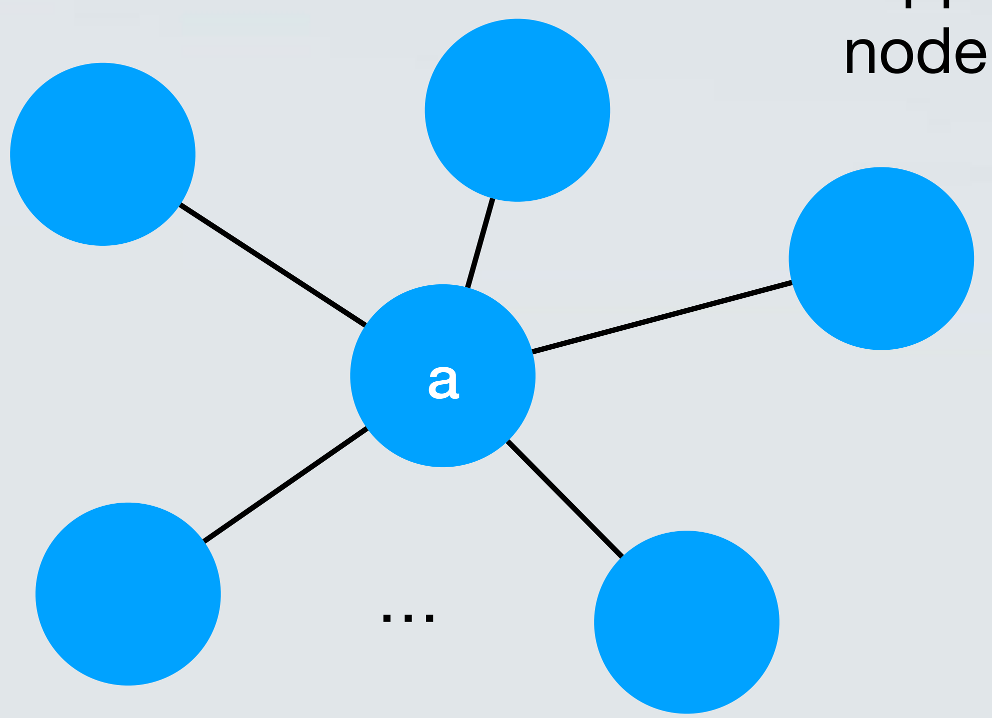
Suppose that the degree of node a was smaller than k



A first observation

- The *maximum degree* in G is at least k.
  - (Why?)

Suppose that the degree of
node a was smaller than k



Is (A, B) a minimum cut?

Where things can go wrong

- Let’s consider the first step of the contraction algorithm.

Where things can go wrong

- Let's consider the first step of the contraction algorithm.
- We will have made a mistake, if an edge e in F was contracted.

Where things can go wrong

- Let's consider the first step of the contraction algorithm.
- We will have made a mistake, if an edge e in F was contracted.
  - When we contract an edge, we irrevocably decide that its endpoints will be in the same "side" of the cut.

Where things can go wrong

- Let's consider the first step of the contraction algorithm.
- We will have made a mistake, if an edge e in F was contracted.
  - When we contract an edge, we irrevocably decide that its endpoints will be in the same "side" of the cut.
  - For an edge e in F, its endpoints lie in different "sides" of the cut.

Where things can go wrong

- Let's consider the first step of the contraction algorithm.
- We will have made a mistake, if an edge e in F was contracted.
  - When we contract an edge, we irrevocably decide that its endpoints will be in the same "side" of the cut.
  - For an edge e in F, its endpoints lie in different "sides" of the cut.
  - If we contract e, then we can't possibly produce the cut (A, B).

Bounding the error probability

- What is the probability that we pick an edge in F?

Bounding the error probability

- What is the probability that we pick an edge in F?
  - F has k edges, we pick uniformly at random from $|E|$ edges.

Bounding the error probability

- What is the probability that we pick an edge in F?
  - F has k edges, we pick uniformly at random from |E| edges.
  - The probability is $k / |E|$.

Bounding the error probability

- What is the probability that we pick an edge in F?
  - F has k edges, we pick uniformly at random from |E| edges.
  - The probability is k / |E|.
  - We want to *upper bound* this quantity.

Bounding the error probability

- What is the probability that we pick an edge in F?
  - F has k edges, we pick uniformly at random from |E| edges.
  - The probability is $k / |E|$.
  - We want to *upper bound* this quantity.
  - We can *lower bound* |E|.

Bounding the error probability

- What is the probability that we pick an edge in F?
  - F has k edges, we pick uniformly at random from |E| edges.
  - The probability is k / |E|.
  - We want to *upper bound* this quantity.
  - We can *lower bound* |E|.
  - Claim: $|E| \geq (kn)/2$. (why?)

Bounding the error probability

- What is the probability that we pick an edge in F?
  - F has k edges, we pick uniformly at random from |E| edges.
  - The probability is k / |E|.
  - We want to *upper bound* this quantity.
  - We can *lower bound* |E|.
  - Claim: $|E| \geq (kn)/2$. (why?)
- The probability that an edge in F is contracted (*in the first round*) is at most *2/n*.

After round *j*

- Suppose that we have gone through *j* rounds and we have **not** contracted any edges in F yet.
- What is the probability that we contract an edge in F now?
- There are *n-j* super-nodes in the graph G'.
- A cut in G' is also a cut in G.
    - The degree of every super-node of G' is again at least k.
    - $|E_{G'}| \geq k(n-j)/2$.
    - The mistake probability is $k / |E_{G'}| = 2/(n-j)$.

Events

- Mistake: Contract an edge in F.
- Event E_j: The algorithm does not make a mistake in round j.
- We have shown:

$$\mathbf{Pr}[E_1] \geq 1 - \frac{2}{n}$$

$$\mathbf{Pr}[E_{j+1} \mid E_1 \cap E_2 \cap \ldots \cap E_j] \geq 1 - \frac{2}{n-j}$$

Events

- Mistake: Contract an edge in F.
- Event E_j: The algorithm does not make a mistake in round j.

Events

- Mistake: Contract an edge in F.
- Event E_j: The algorithm does not make a mistake in round j.
- When is our algorithm successful?

Events

- Mistake: Contract an edge in F.
- Event E_j: The algorithm does not make a mistake in round *j*.
- When is our algorithm successful?
  - When it has not made a mistake in any round.

Events

- Mistake: Contract an edge in F.
- Event E_j: The algorithm does not make a mistake in round j.
- When is our algorithm successful?
  - When it has not made a mistake in any round.
  - What is the probability of that happening?

Events

- Mistake: Contract an edge in F.
- Event E_j: The algorithm does not make a mistake in round j.
- When is our algorithm successful?
  - When it has not made a mistake in any round.
  - What is the probability of that happening?

$$\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_j]$$

Calculating

Calculating

We want to find: $\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_{n-2}]$

Calculating

We want to find: $\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_{n-2}]$

We know: $\mathbf{Pr}[E_1] \geq 1 - \frac{2}{n}$

$$\mathbf{Pr}[E_{j+1} \,|\, E_1 \cap E_2 \cap \ldots \cap E_j] \geq 1 - \frac{2}{n-j}$$

Calculating

We want to find: $\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_{n-2}]$

We know: $\mathbf{Pr}[E_1] \geq 1 - \frac{2}{n}$

$$\mathbf{Pr}[E_{j+1} \mid E_1 \cap E_2 \cap \ldots \cap E_j] \geq 1 - \frac{2}{n-j}$$

By the conditional probability formula:

Calculating

We want to find: $\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_{n-2}]$

We know: $\mathbf{Pr}[E_1] \geq 1 - \frac{2}{n}$

$$\mathbf{Pr}[E_{j+1} \mid E_1 \cap E_2 \cap \ldots \cap E_j] \geq 1 - \frac{2}{n-j}$$

By the conditional probability formula:

$$\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_{n-2}] =$$

$$\mathbf{Pr}[E_1] \cdot \mathbf{Pr}[E_2 \mid E_1] \;\ldots\; \mathbf{Pr}[E_{j+1} \mid E_1 \cap E_2 \cap \ldots \cap E_j] \;\ldots\; \mathbf{Pr}[E_{n-2} \mid E_1 \cap E_2 \cap \ldots \cap E_{n-3}]$$

Calculating

We want to find: $\textbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_{n-2}]$

We know: $\textbf{Pr}[E_1] \geq 1 - \frac{2}{n}$

$$\textbf{Pr}[E_{j+1} | E_1 \cap E_2 \cap \ldots \cap E_j] \geq 1 - \frac{2}{n-j}$$

By the conditional probability formula:

$$\textbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_{n-2}] =$$

$$\textbf{Pr}[E_1] \cdot \textbf{Pr}[E_2 | E_1] \;\ldots\; \textbf{Pr}[E_{j+1} | E_1 \cap E_2 \cap \ldots \cap E_j] \;\ldots\; \textbf{Pr}[E_{n-2} | E_1 \cap E_2 \cap \ldots \cap E_{n-3}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\ldots\left(1 - \frac{2}{n-j}\right)\ldots\left(1 - \frac{2}{3}\right)$$

Calculating

By the conditional probability formula:

$$\textbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_j] =$$

$$\textbf{Pr}[E_1] \cdot \textbf{Pr}[E_2 \,|\, E_1] \;\; \ldots \;\; \textbf{Pr}[E_{j+1} \,|\, E_1 \cap E_2 \cap \ldots \cap E_j] \;\; \ldots \;\; \textbf{Pr}[E_{n-2} \,|\, E_1 \cap E_2 \cap \ldots \cap E_{n-3}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\ldots\left(1 - \frac{2}{n-j}\right)\ldots\left(1 - \frac{2}{3}\right)$$

Calculating

By the conditional probability formula:

$$\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_j] =$$

$$\mathbf{Pr}[E_1] \cdot \mathbf{Pr}[E_2 \mid E_1] \;\ldots\; \mathbf{Pr}[E_{j+1} \mid E_1 \cap E_2 \cap \ldots \cap E_j] \;\ldots\; \mathbf{Pr}[E_{n-2} \mid E_1 \cap E_2 \cap \ldots \cap E_{n-3}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\ldots\left(1 - \frac{2}{n-j}\right)\ldots\left(1 - \frac{2}{3}\right)$$

$$\geq \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\ldots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right)$$

Calculating

By the conditional probability formula:

$$\mathbf{Pr}[E_1 \cap E_2 \cap \ldots \cap E_j] =$$

$$\mathbf{Pr}[E_1] \cdot \mathbf{Pr}[E_2 \,|\, E_1] \;\; \ldots \;\; \mathbf{Pr}[E_{j+1} \,|\, E_1 \cap E_2 \cap \ldots \cap E_j] \;\; \ldots \;\; \mathbf{Pr}[E_{n-2} \,|\, E_1 \cap E_2 \cap \ldots \cap E_{n-3}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\ldots\left(1 - \frac{2}{n-j}\right)\ldots\left(1 - \frac{2}{3}\right)$$

$$\geq \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\ldots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right)$$

$$= \frac{2}{n(n-1)} = \binom{n}{2}^{-1}$$

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

$$1 - \frac{1}{\binom{n}{2}} = 1 - \frac{2}{n(n-1)}$$

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.
- It will produce the correct outcome *with high probability*.
    - Sometimes it might make a mistake!
    - But not too often.

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.
- It will produce the correct outcome *with high probability*.
    - Sometimes it might make a mistake!
    - But not too often.

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

$$1 - \frac{1}{\binom{n}{2}} = 1 - \frac{2}{n(n-1)}$$

Application

- Suppose that we repeat an experiment multiple times, and each time the probability of success is $p > 0$.
    - e.g., compute a minimum cut in a graph.

Success Amplification

- Run the algorithm independently X times.
- The probability that it fails is equal to

  the probability that it fails the 1st time x
  the probability that it fails the 2nd time x
  ...
  the probability that it fails the Xth time.

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

- If we run the algorithm independently Binom(n, k) times, the probability of error becomes

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

$$1 - \frac{1}{\binom{n}{2}} = 1 - \frac{2}{n(n-1)}$$

- If we run the algorithm independently Binom(n, k) times, the probability of error becomes

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

$$1 - \frac{1}{\binom{n}{2}} = 1 - \frac{2}{n(n-1)}$$

- If we run the algorithm independently Binom(n, k) times, the probability of error becomes

$$\left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}} \leq \frac{1}{e}$$

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

- If we run the algorithm independently Binom(n, k) ln e times, the probability of error becomes

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

$$1 - \frac{1}{\binom{n}{2}} = 1 - \frac{2}{n(n-1)}$$

- If we run the algorithm independently Binom(n, k) ln e times, the probability of error becomes

Probability of failure

- The contraction algorithm fails to find a global minimum s-t cut with probability at most

$$1 - \frac{1}{\binom{n}{2}} = 1 - \frac{2}{n(n-1)}$$

- If we run the algorithm independently Binom(n, k) ln e times, the probability of error becomes

$$\left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2} \ln e} \leq \frac{1}{n}$$

Generally

- We can run the algorithm independently a number of times.
- This will decrease the error probability.
- This will increase the running time.
- There is a trade-off between the two.

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.
- It will produce the correct outcome *with high probability*.
  - Sometimes it might make a mistake!
  - But not too often.

Is there a simpler solution?

- We will use a randomised algorithm to solve the problem.
- The algorithm will be faster and simpler.
- It will produce the correct outcome *with high probability*.
  - Sometimes it might make a mistake!
  - But not too often.

Faster and simpler

Faster and simpler

- Definitely simpler.

Faster and simpler

- Definitely simpler.
- To get high success probability, we need a lot of repetitions, so does not seem faster.

Faster and simpler

- Definitely simpler.
- To get high success probability, we need a lot of repetitions, so does not seem faster.
  - One can do clever optimisations to the way in which multiple runs are performed to improve the running time considerably.