Probability & Computing

Probability Amplification

The Segmentation Problem

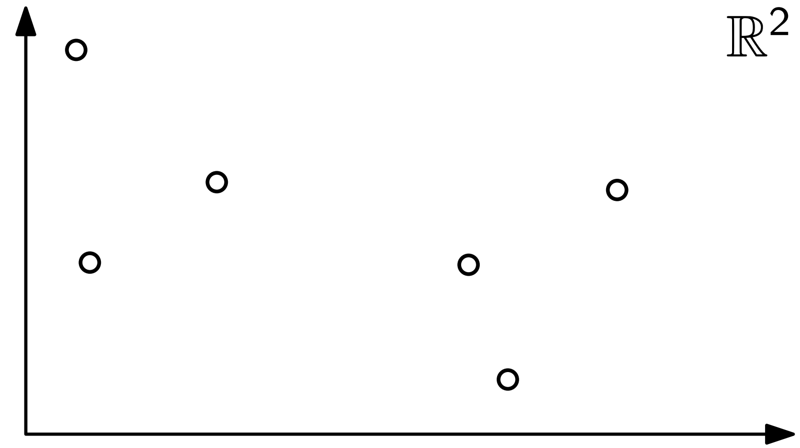
Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma\colon P \times P \mapsto \mathbb{R}_+$

The Segmentation Problem

Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma\colon P \times P \mapsto \mathbb{R}_+$



Example

- six points in $\mathbb{R}^2$
- σ is the inversed Euclidean distance

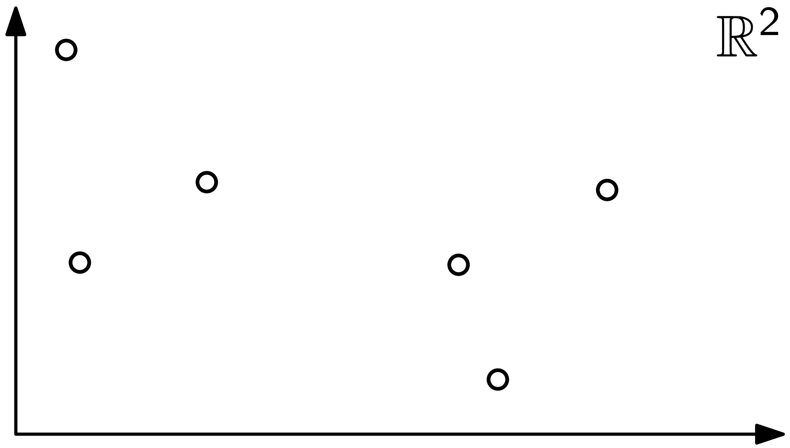
The Segmentation Problem

Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma : P \times P \mapsto \mathbb{R}_+$

Output: $P_1, \ldots, P_k$ such that

- Points within a P_i have high similarity
- Points in distinct P_i, P_j have low similarity



Example

- six points in $\mathbb{R}^2$
- σ is the inversed Euclidean distance
- segment into two sets

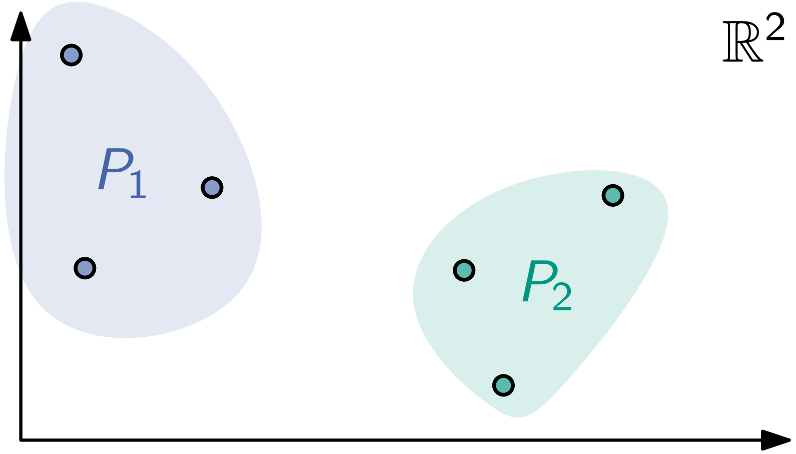
The Segmentation Problem

Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma\colon P \times P \mapsto \mathbb{R}_+$

Output: $P_1, \ldots, P_k$ such that

- Points within a P_i have high similarity
- Points in distinct P_i, P_j have low similarity



Example

- six points in $\mathbb{R}^2$
- σ is the inversed Euclidean distance
- segment into two sets

The Segmentation Problem

Karlsruhe Institute of Technology

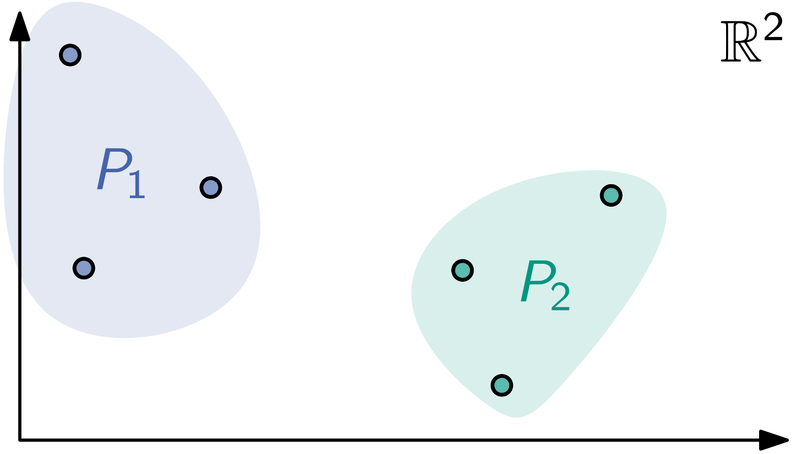
Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma : P \times P \mapsto \mathbb{R}_+$

Output: $P_1, \ldots, P_k$ such that

- Points within a P_i have high similarity
- Points in distinct P_i, P_j have low similarity

Applications: Compression, medical diagnosis, etc.



Example

- six points in $\mathbb{R}^2$
- σ is the inversed Euclidean distance
- segment into two sets

The Segmentation Problem

Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma : P \times P \mapsto \mathbb{R}_+$

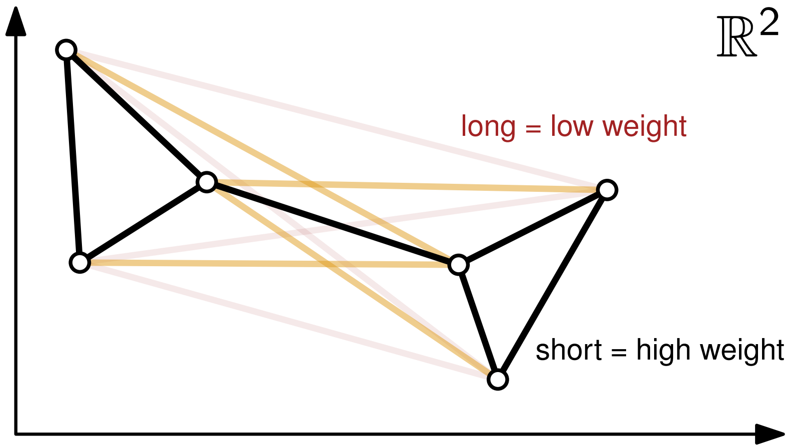
Output: $P_1, \ldots, P_k$ such that

- Points within a P_i have high similarity
- Points in distinct P_i, P_j have low similarity

Applications: Compression, medical diagnosis, etc.

Approach: Model as graph

- Each point is a node
- Edges between all node pairs, with the weight given by the similarity of the two nodes



Example

- six points in $\mathbb{R}^2$
- σ is the inversed Euclidean distance
- segment into two sets

The Segmentation Problem

Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma \colon P \times P \mapsto \mathbb{R}_+$

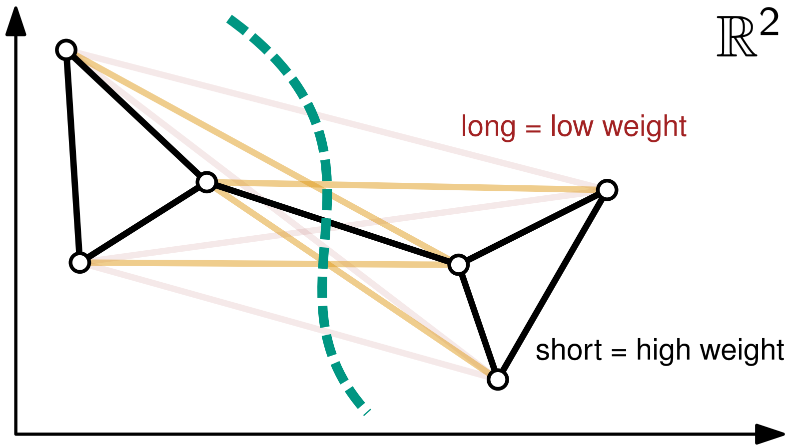
Output: $P_1, \ldots, P_k$ such that

- Points within a P_i have high similarity
- Points in distinct P_i, P_j have low similarity

Applications: Compression, medical diagnosis, etc.

Approach: Model as graph

- Each point is a node
- Edges between all node pairs, with the weight given by the similarity of the two nodes
- Find *cut-set* (edges to remove) of minimal weight such that the graph decomposes into k components.



Example

- six points in $\mathbb{R}^2$
- σ is the inversed Euclidean distance
- segment into two sets

The Segmentation Problem

Input

- Set P of points in a feature space (e.g., $\mathbb{R}^d$)
- Similarity measure $\sigma\colon P \times P \mapsto \mathbb{R}_+$

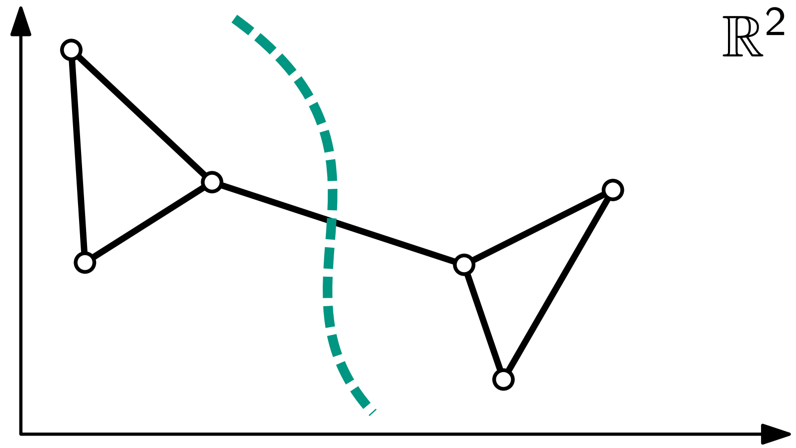
Output: $P_1, \ldots, P_k$ such that

- Points within a P_i have high similarity
- Points in distinct P_i, P_j have low similarity

Applications: Compression, medical diagnosis, etc.

Approach: Model as graph

- Each point is a node
- Edges between all node pairs, with the weight given by the similarity of the two nodes
- Find *cut-set* (edges to remove) of minimal weight such that the graph decomposes into k components.



Example

- six points in $\mathbb{R}^2$
- σ is the inversed Euclidean distance
- segment into two sets

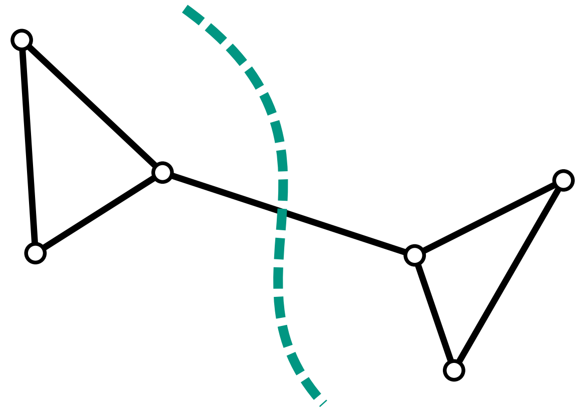
Today

$k = 2$ and $\sigma\colon P \times P \mapsto \{0, 1\}$

The Edge-Connectivity Problem

Cuts

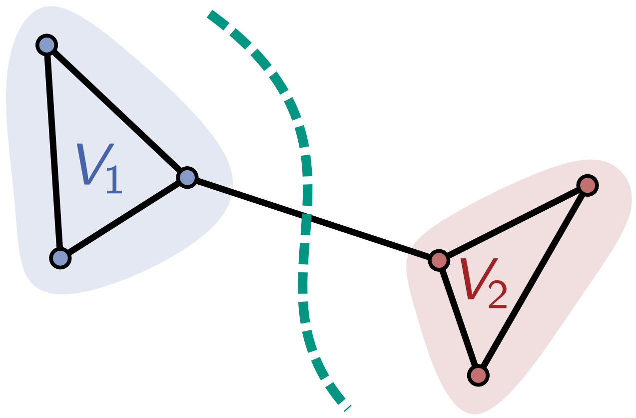
- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$.
  (in general one can consider more than two parts)



The Edge-Connectivity Problem

Cuts

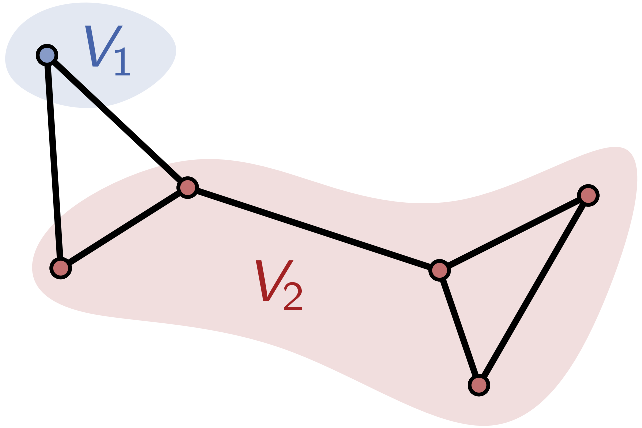
- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$.
  (in general one can consider more than two parts)



The Edge-Connectivity Problem

Cuts

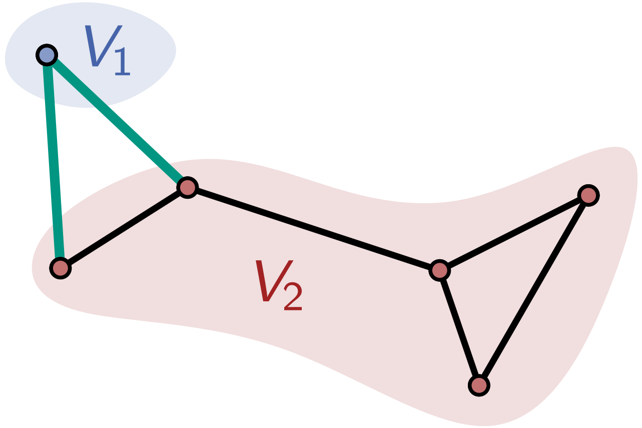
- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$.
  (in general one can consider more than two parts)



The Edge-Connectivity Problem

Cuts

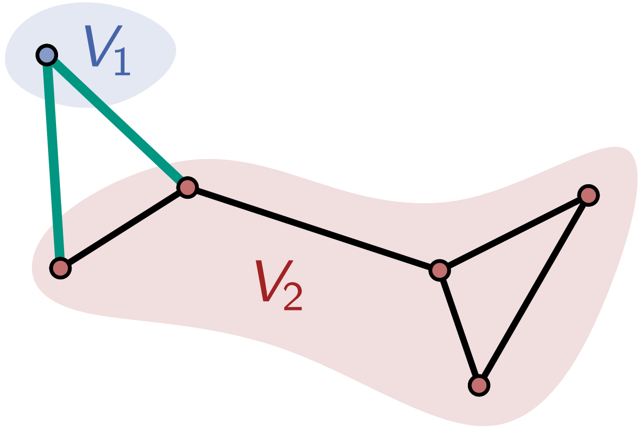
- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$.
  (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2



The Edge-Connectivity Problem

Cuts

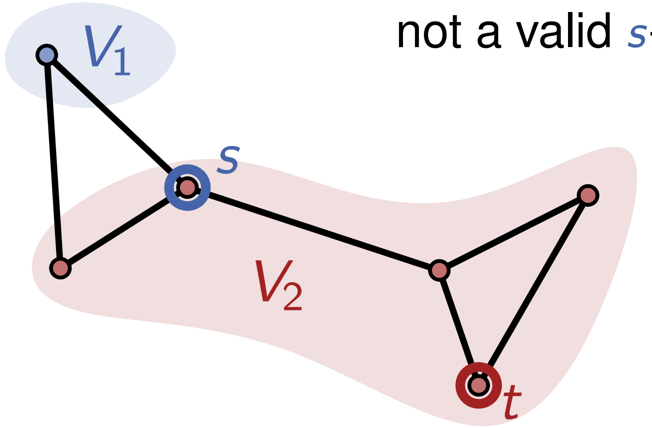
- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)



The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)



not a valid s-t-cut

Excursion: Cuts with Terminals

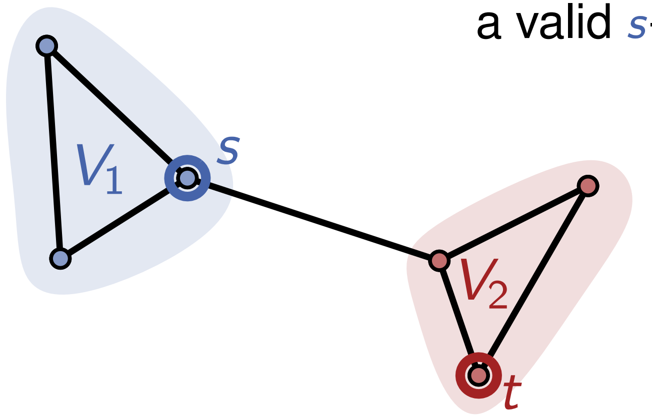
- each part contains exectly one of a specified vertex set

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

a valid s-t-cut



Excursion: Cuts with Terminals

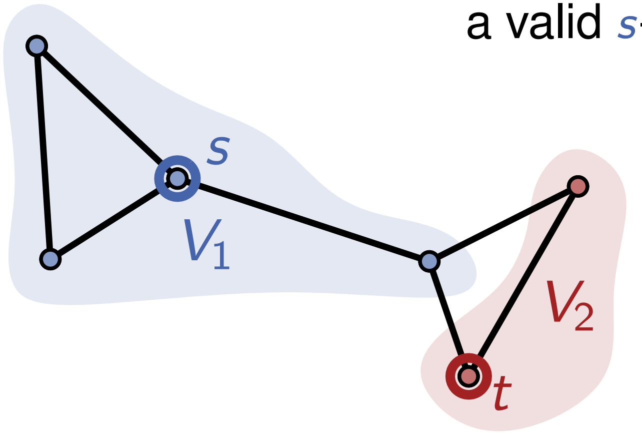
- each part contains exectly one of a specified vertex set

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

a valid s-t-cut



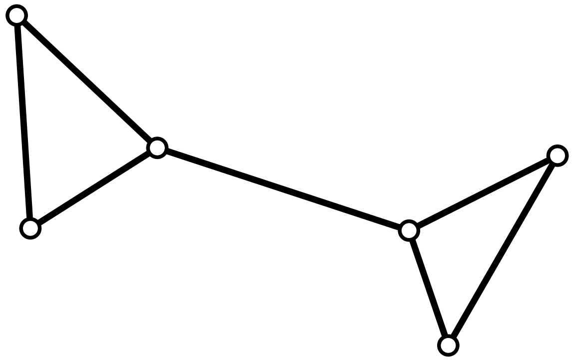
Excursion: Cuts with Terminals

- each part contains exectly one of a specified vertex set

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)



Excursion: Cuts with Terminals

- each part contains exectly one of a specified vertex set

k-Edge-Connectivity

- *k-edge-connected*: a minimum cut has weight at least k

(we cannot disconnect the graph by removing less than k edges)

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

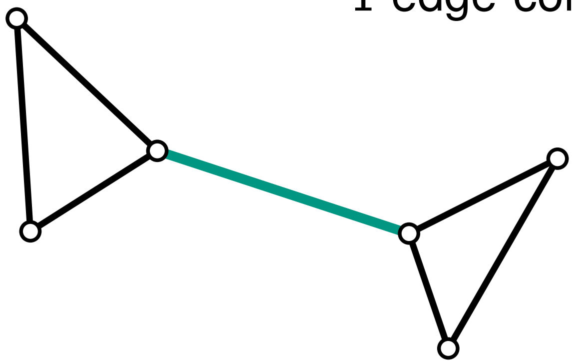
Excursion: Cuts with Terminals

- each part contains exectly one of a specified vertex set

k-Edge-Connectivity

- *k-edge-connected*: a minimum cut has weight at least k (we cannot disconnect the graph by removing less than k edges)

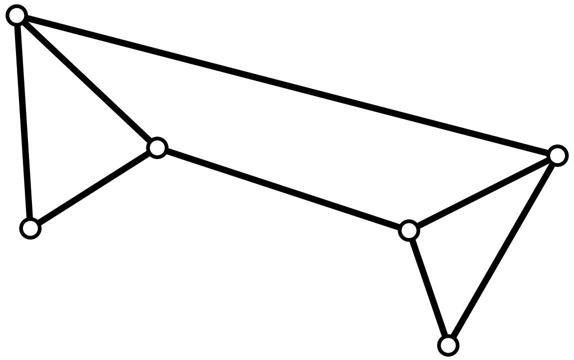
1-edge-connected



The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$.
  (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set
  (or sum of weights in a weighted graph)



Excursion: Cuts with Terminals

- each part contains exectly one of a specified vertex set

k-Edge-Connectivity

- *k-edge-connected*: a minimum cut has weight at least k
  (we cannot disconnect the graph by removing less than k edges)

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

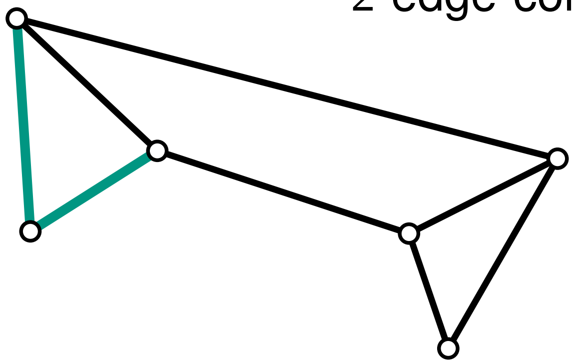
Excursion: Cuts with Terminals

- each part contains exectly one of a specified vertex set

k-Edge-Connectivity

- k*-edge-connected*: a minimum cut has weight at least k

(we cannot disconnect the graph by removing less than k edges)

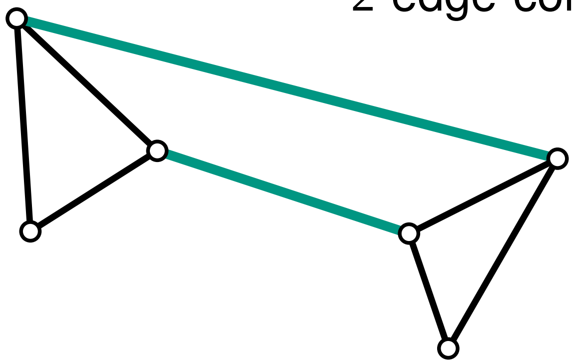


The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

2-edge-connected



Excursion: Cuts with Terminals

- each part contains exectly one of a specified vertex set

k-Edge-Connectivity

- *k-edge-connected*: a minimum cut has weight at least k

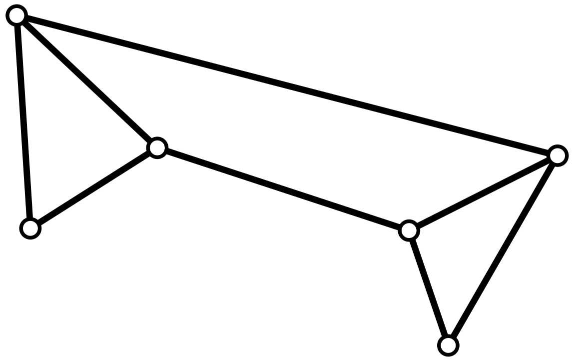
(we cannot disconnect the graph by removing less than k edges)

The Edge-Connectivity Problem

Karlsruhe Institute of Technology

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)



Excursion: Cuts with Terminals

- each part contains exactly one of a specified vertex set

k-Edge-Connectivity

- *k-edge-connected*: a minimum cut has weight at least k

(we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

Excursion: Cuts with Terminals

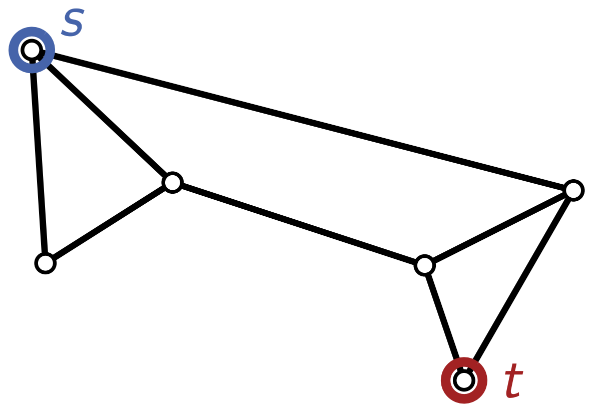
- each part contains exactly one of a specified vertex set

k-Edge-Connectivity

- k*-edge-connected*: a minimum cut has weight at least k (we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)



Excursion: Flows

- given source s and target t
- assign *flow* to edges s.t.
  - in-flow $=$ out-flow for all vertices (not s and t)
  - flow of an edge bounded by edge-capacity (here: ≤ 1)
  - flow in t is maximized

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

Excursion: Cuts with Terminals

- each part contains exactly one of a specified vertex set

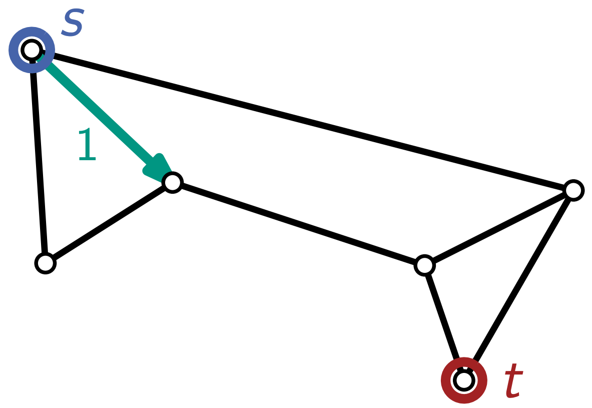
k-Edge-Connectivity

- k*-edge-connected*: a minimum cut has weight at least k

(we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)



Excursion: Flows

- given source s and target t
- assign *flow* to edges s.t.
  - in-flow $=$ out-flow for all vertices (not s and t)
  - flow of an edge bounded by edge-capacity (here: ≤ 1)
  - flow in t is maximized

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

Excursion: Cuts with Terminals

- each part contains exactly one of a specified vertex set

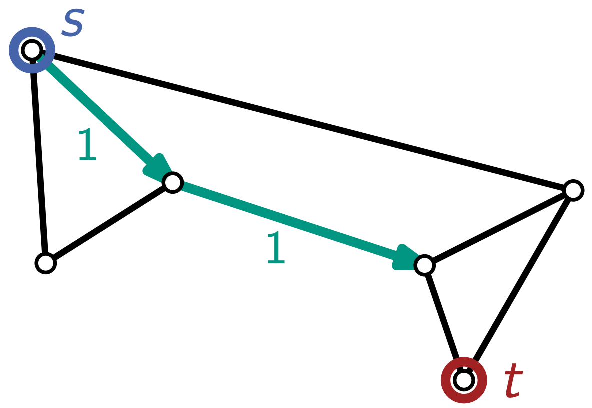
k-Edge-Connectivity

- k*-edge-connected*: a minimum cut has weight at least k

(we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)



Excursion: Flows

- given source s and target t
- assign *flow* to edges s.t.
  - in-flow $=$ out-flow for all vertices (not s and t)
  - flow of an edge bounded by edge-capacity (here: ≤ 1)
  - flow in t is maximized

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

Excursion: Cuts with Terminals

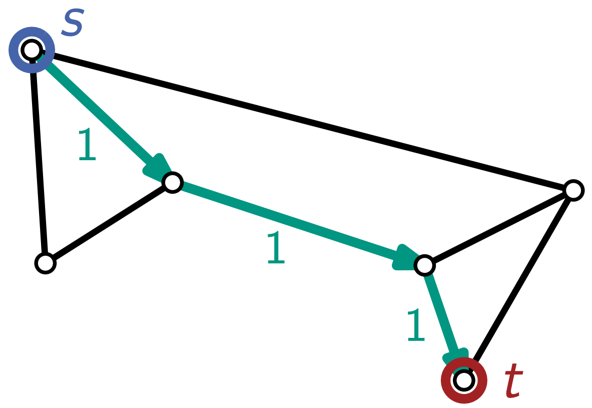
- each part contains exactly one of a specified vertex set

k-Edge-Connectivity

- k*-edge-connected*: a minimum cut has weight at least k (we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)



Excursion: Flows

- given source s and target t
- assign *flow* to edges s.t.
  - in-flow = out-flow for all vertices (not s and t)
  - flow of an edge bounded by edge-capacity (here: ≤ 1)
  - flow in t is maximized

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

Excursion: Cuts with Terminals

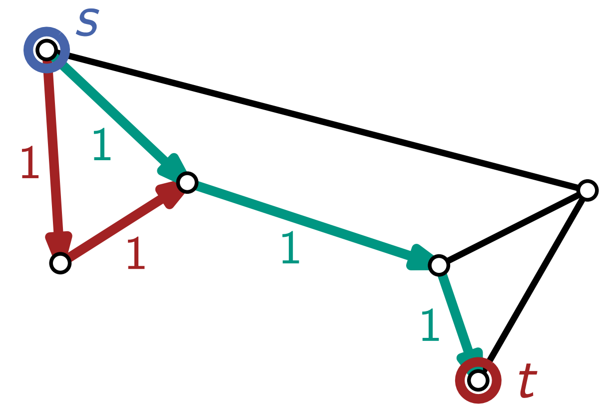
- each part contains exactly one of a specified vertex set

k-Edge-Connectivity

- k*-edge-connected*: a minimum cut has weight at least k (we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)



Excursion: Flows

- given source s and target t
- assign *flow* to edges s.t.
  - in-flow $=$ out-flow for all vertices (not s and t)
  - flow of an edge bounded by edge-capacity (here: ≤ 1)
  - flow in t is maximized

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

Excursion: Cuts with Terminals

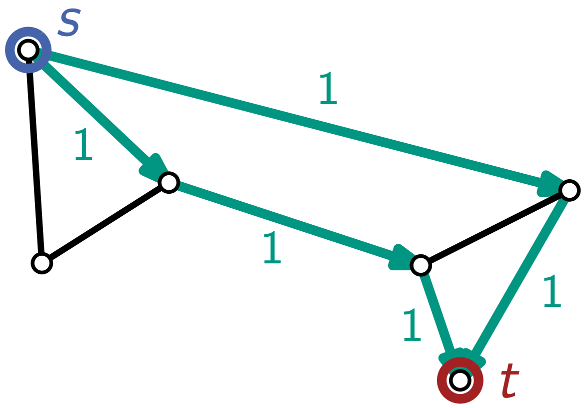
- each part contains exactly one of a specified vertex set

k-Edge-Connectivity

- k*-edge-connected*: a minimum cut has weight at least k (we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)



Excursion: Flows

- given source s and target t
- assign *flow* to edges s.t.
  - in-flow $=$ out-flow for all vertices (not s and t)
  - flow of an edge bounded by edge-capacity (here: ≤ 1)
  - flow in t is maximized

The Edge-Connectivity Problem

Cuts

- $G = (V, E)$ an unweighted, undirected, connected graph
- *Cut*: partition of V into parts V_1, V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. (in general one can consider more than two parts)
- *Cut-set*: set of edges with one endpoint in V_1 and the other in V_2
- *Weight*: size of the cut-set (or sum of weights in a weighted graph)

Excursion: Cuts with Terminals

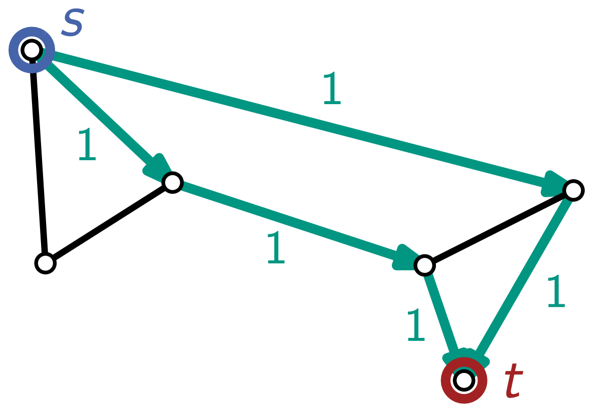
- each part contains exectly one of a specified vertex set

k-Edge-Connectivity

- *k-edge-connected*: a minimum cut has weight at least k (we cannot disconnect the graph by removing less than k edges)

Edge-Connectivity

- max. k such that G is k-edge-connected (exactly the weight of a min-cut)



Excursion: Flows

- given source s and target t
- assign *flow* to edges s.t.
  - in-flow $=$ out-flow for all vertices (not s and t)
  - flow of an edge bounded by edge-capacity (here: ≤ 1)
  - flow in t is maximized

Thm. Max-Flow = Min-Cut.

Deterministic Algorithms for Edge-Connectivity

Flow-based

- Compute max-flow between all vertex pairs $\rightarrow O(n^2 \cdot T_{\text{max-flow}})$

Deterministic Algorithms for Edge-Connectivity

Flow-based

- Compute max-flow between all vertex pairs $\rightarrow O(n^2 \cdot \overbrace{T_{\text{max-flow}}}^{O(nm)}) \subseteq O(n^3 m)$

"Max flows in O(nm) time, or better", Orlin, STOC'13

Deterministic Algorithms for Edge-Connectivity

Flow-based

"Max flows in O(nm) time, or better", Orlin, STOC'13

- Compute max-flow between all vertex pairs $\rightarrow O(n^2 \cdot \overbrace{T_{\text{max-flow}}}^{O(nm)}) \subseteq O(n^3 m)$
- Compute max-flow between v and all others $\rightarrow O(n \cdot T_{\text{max-flow}}) \subseteq O(n^2 m) \rightarrow \Omega(n^3)$

  (if a cut of size k exists, it has to cut v from some vertex)

Deterministic Algorithms for Edge-Connectivity

Karlsruhe Institute of Technology

Flow-based

"Max flows in O(nm) time, or better", Orlin, STOC'13

- Compute max-flow between all vertex pairs $\rightarrow O(n^2 \cdot \overbrace{T_{\text{max-flow}}}^{O(nm)}) \subseteq O(n^3 m)$
- Compute max-flow between v and all others $\rightarrow O(n \cdot T_{\text{max-flow}}) \subseteq O(n^2 m) \rightarrow \Omega(n^3)$

(if a cut of size k exists, it has to cut v from some vertex)

Matroid-based

"A Matroid Approach to Finding Edge Connectivity and Packing Arborescences", Gabow, JCSS, 1995

- Involved technique based on the fact that min-cut $=$ max. number of dijsoint, directed spanning trees $\rightarrow O(m + k^2 n \log(n/k))$

Deterministic Algorithms for Edge-Connectivity

Flow-based

"Max flows in O(nm) time, or better", Orlin, STOC'13

- Compute max-flow between all vertex pairs $\rightarrow O(n^2 \cdot \underbrace{T_{\text{max-flow}}}_{O(nm)}) \subseteq O(n^3 m)$
- Compute max-flow between v and all others $\rightarrow O(n \cdot T_{\text{max-flow}}) \subseteq O(n^2 m) \rightarrow \Omega(n^3)$

(if a cut of size k exists, it has to cut v from some vertex)

Matroid-based

"A Matroid Approach to Finding Edge Connectivity and Packing Arborescences", Gabow, JCSS, 1995

- Involved technique based on the fact that min-cut $=$ max. number of dijsoint, directed spanning trees $\rightarrow O(m + k^2 n \log(n/k))$
- Good if k is small but still $\Omega(n^3)$ in the worst case

Deterministic Algorithms for Edge-Connectivity

Flow-based

"Max flows in O(nm) time, or better", Orlin, STOC'13

- Compute max-flow between all vertex pairs $\rightarrow O(n^2 \cdot \overbrace{T_{\text{max-flow}}}^{O(nm)}) \subseteq O(n^3 m)$
- Compute max-flow between v and all others $\rightarrow O(n \cdot T_{\text{max-flow}}) \subseteq O(n^2 m) \rightarrow \Omega(n^3)$

(if a cut of size k exists, it has to cut v from some vertex)

Matroid-based

"A Matroid Approach to Finding Edge Connectivity and Packing Arborescences", Gabow, JCSS, 1995

- Involved technique based on the fact that min-cut $=$ max. number of dijsoint, directed spanning trees $\rightarrow O(m + k^2 n \log(n/k))$
- Good if k is small but still $\Omega(n^3)$ in the worst case

Contraction-based

"A simple min-cut algorithm", Stoer & Wagner, JACM, 1997

- Iteratively pick two vertices (in a smart way) and compare the min-cuts where they are / are not in the same part $\rightarrow O(mn + n^2 \log(n)) \rightarrow \Omega(n^3)$

Deterministic Algorithms for Edge-Connectivity

Karlsruhe Institute of Technology

Flow-based

“Max flows in O(nm) time, or better”, Orlin, STOC’13

- Compute max-flow between all vertex pairs $\rightarrow O(n^2 \cdot \underbrace{T_{\text{max-flow}}}_{O(nm)}) \subseteq O(n^3 m)$
- Compute max-flow between v and all others $\rightarrow O(n \cdot T_{\text{max-flow}}) \subseteq O(n^2 m) \rightarrow \Omega(n^3)$

(if a cut of size k exists, it has to cut v from some vertex)

Matroid-based

“A Matroid Approach to Finding Edge Connectivity and Packing Arborescences”, Gabow, JCSS, 1995

- Involved technique based on the fact that min-cut $=$ max. number of dijsoint, directed spanning trees $\rightarrow O(m + k^2 n \log(n/k))$
- Good if k is small but still $\Omega(n^3)$ in the worst case

Contraction-based

“A simple min-cut algorithm”, Stoer & Wagner, JACM, 1997

- Iteratively pick two vertices (in a smart way) and compare the min-cuts where they are / are not in the same part $\rightarrow O(mn + n^2 \log(n)) \rightarrow \Omega(n^3)$

Enter: The Power of Randomness!

A Simple(?) Randomized Algorithm

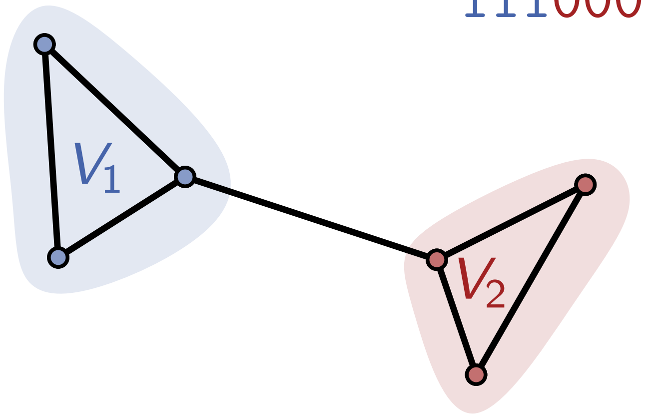
Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

111000

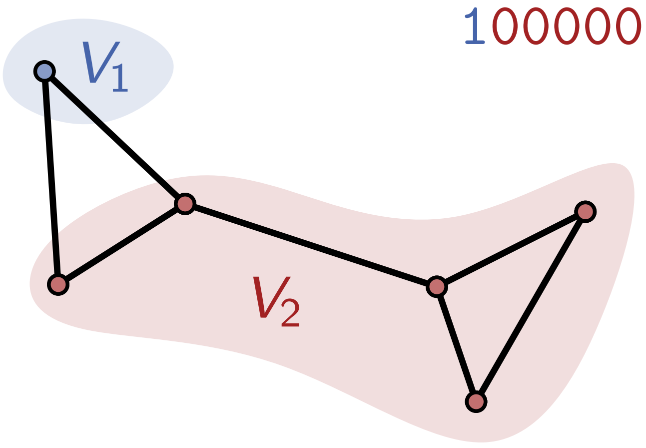
- Number of possible assignments of n nodes to 2 parts 2^n



A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

- Number of possible assignments of n nodes to 2 parts $\to 2^n$

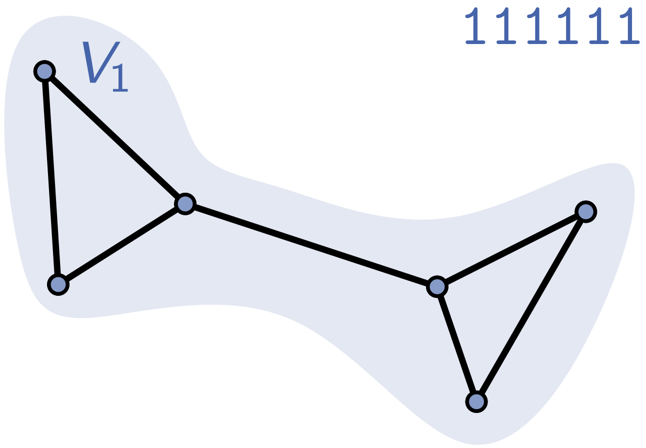


A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$2^n - 2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts

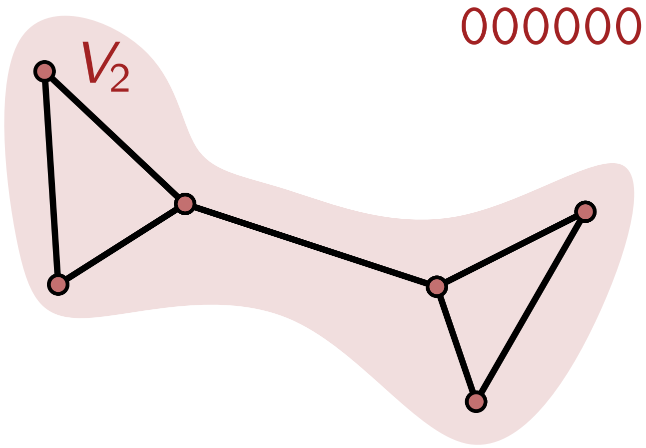


A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$2^n - 2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts



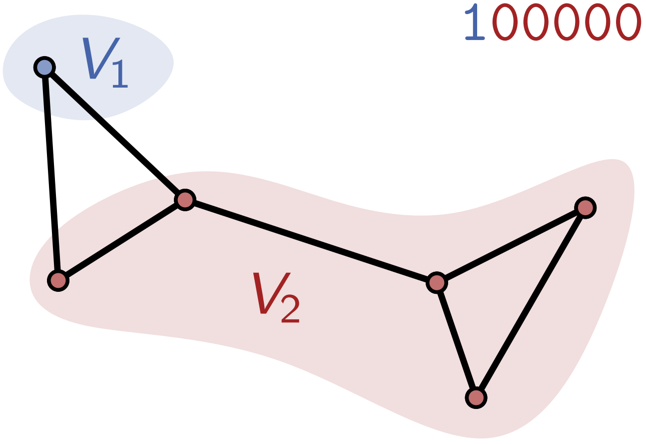
A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$\left(2^n - 2\right) / 2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition

100000

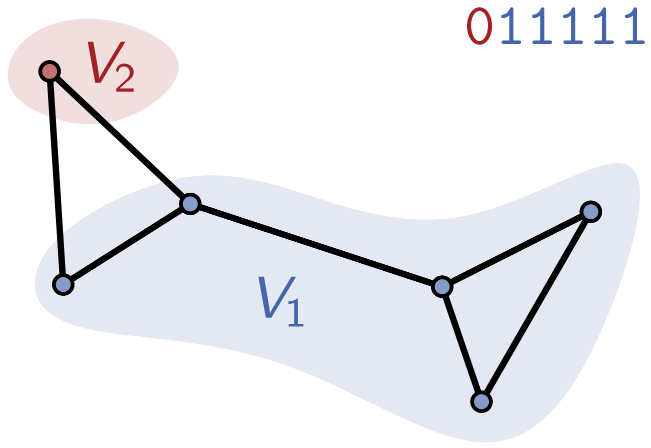


A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$(2^n - 2)/2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition

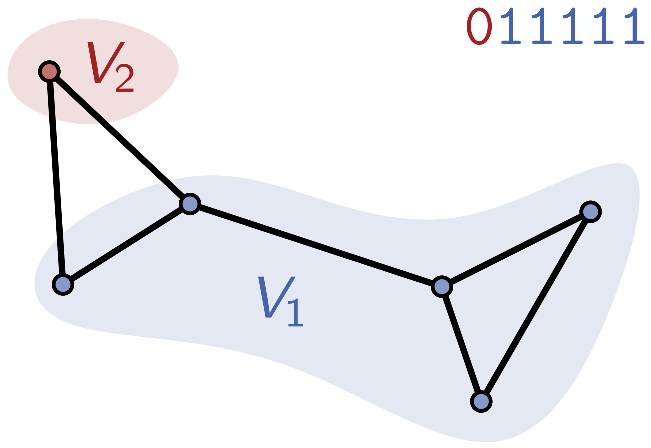


A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$$(2^n - 2)/2$$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition



Algorithm: Simple(?) Randomized Cut

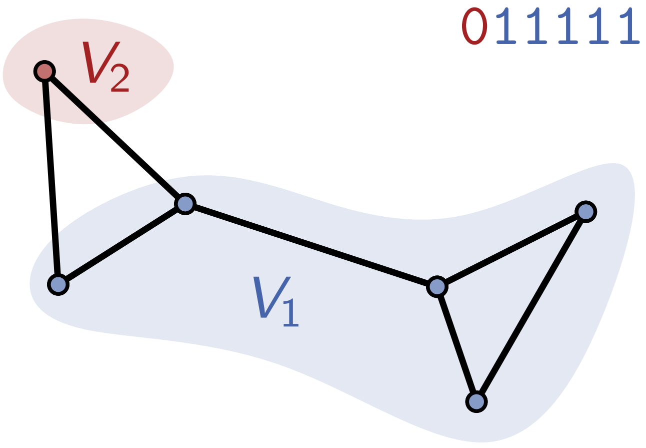
- Simple idea: choose a cut at random among all possible cuts and return it.

A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$\left(2^n - 2\right)/2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition



Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

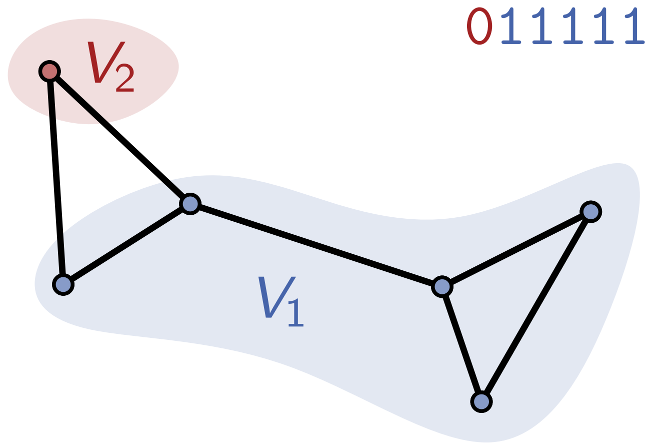
What do we mean?
What distribution?

A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$\left(2^n - 2\right)/2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition



Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

  What do we mean?
  What distribution?

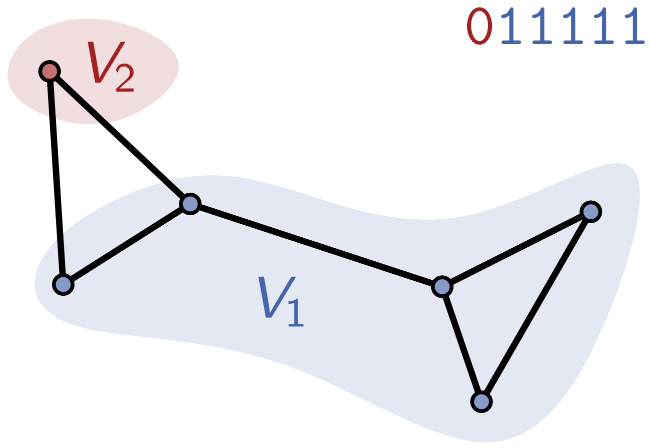
- Uniform distribution: We do not want to potentially favor non-minimum cuts

A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$$\left(2^n - 2\right)/2$$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition



Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

What do we mean?
What distribution?

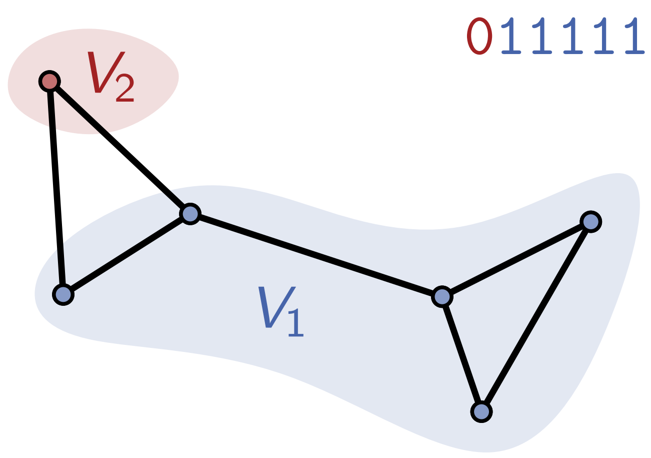
- Uniform distribution: We do not want to potentially favor non-minimum cuts
- Problem: How do we choose a cut *uniformly* at random?

A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$\left(2^n - 2\right)/2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition



Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

What do we mean?
What distribution?

- Uniform distribution: We do not want to potentially favor non-minimum cuts
- Problem: How do we choose a cut *uniformly* at random?
  - Represent cut using bit-string

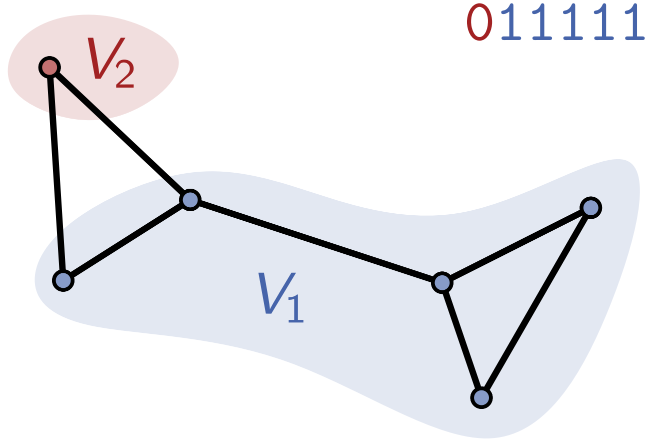
A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$$\left(2^n - 2\right)/2$$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition

011111



Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

  What do we mean?
  What distribution?

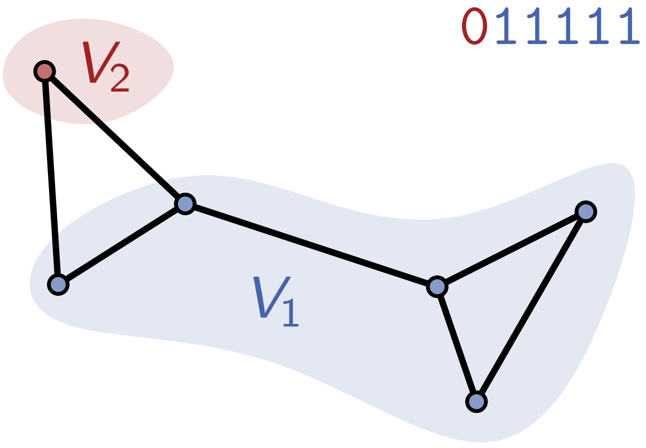
- Uniform distribution: We do not want to potentially favor non-minimum cuts
- Problem: How do we choose a cut *uniformly* at random?
  - Represent cut using bit-string
  - How can we choose a unfiorm random bit-string *while avoiding* `11...1` and `00...0`?

A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$\left(2^n - 2\right)/2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition



Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

What do we mean?
What distribution?

- Uniform distribution: We do not want to potentially favor non-minimum cuts
- Problem: How do we choose a cut *uniformly* at random?
  - Represent cut using bit-string
  - How can we choose a unfiorm random bit-string *while avoiding* `11...1` and `00...0`?

n random bits? $\rightarrow$ does not avoid `11...1` and `00...0`

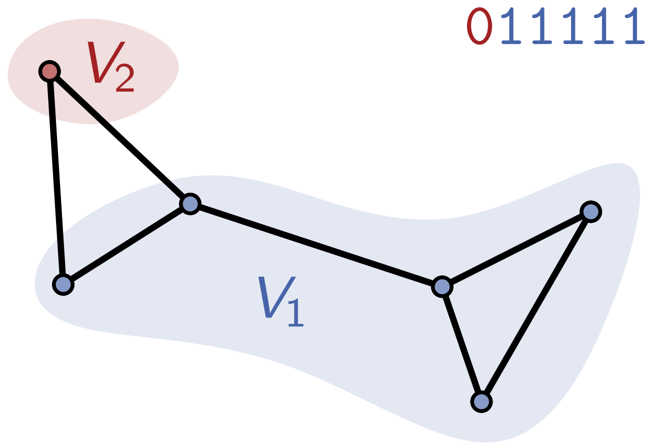
A Simple(?) Randomized Algorithm

Karlsruhe Institute of Technology

> **Observation**: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$(2^n - 2)/2$

- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition



Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

  What do we mean?
  What distribution?

- Uniform distribution: We do not want to potentially favor non-minimum cuts
- Problem: How do we choose a cut *uniformly* at random?
  - Represent cut using bit-string
  - How can we choose a unfiorm random bit-string *while avoiding* `11...1` and `00...0`?

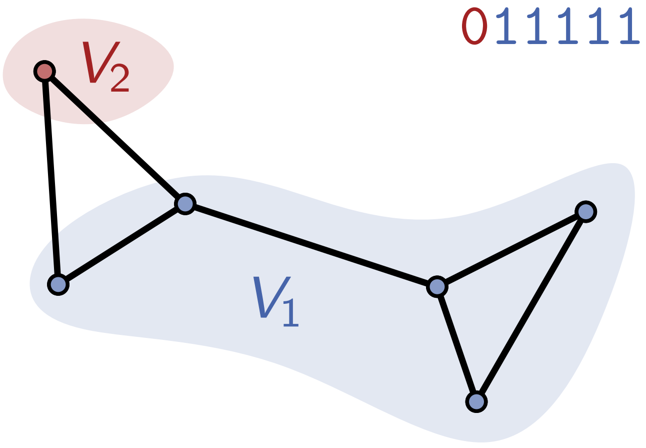
    n random bits? $\rightarrow$ does not avoid `11...1` and `00...0`

    rejection sampling? running time not deterministic (though probably what you'd do in practice)

A Simple(?) Randomized Algorithm

Observation: There are $2^{n-1} - 1$ cuts in a graph with n nodes.

$(2^n - 2)/2$



- Number of possible assignments of n nodes to 2 parts
- Partitions with empty parts that do not represent cuts
- Swapping parts does not yield a new partition

Algorithm: Simple(?) Randomized Cut

- Simple idea: choose a cut at random among all possible cuts and return it.

What do we mean?
What distribution?

- Uniform distribution: We do not want to potentially favor non-minimum cuts
- Problem: How do we choose a cut *uniformly* at random?
  - Represent cut using bit-string
  - How can we choose a unfiorm random bit-string *while avoiding* `11...1` and `00...0`?

n random bits? $\rightarrow$ does not avoid `11...1` and `00...0`

random number from $\{1, \ldots, 2^n - 2\}$? $\rightarrow$ exponential in input size

rejection sampling? running time not deterministic (though probably what you'd do in practice)

Excursion: Uniform Non-Identical Bit Strings

`[For educational purposes only!]`

- **Goal**: Choose uniformly at random from the length n bit-strings that are not $\texttt{0}^n$ or $\texttt{1}^n$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not $\texttt{0}^n$ or $\texttt{1}^n$

Assumptions: We can sample . . .

- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not $\texttt{0}^n$ or $\texttt{1}^n$
- Number of valid bit-strings:

$$2^n - 2$$

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

$n = 4$

$$2^n - 2 \left\{ \begin{array}{l} \texttt{1000} \\ \texttt{0100} \\ \texttt{0010} \\ \texttt{0001} \\ \texttt{1100} \\ \texttt{1010} \\ \texttt{1001} \\ \texttt{0110} \\ \texttt{0101} \\ \texttt{0011} \\ \texttt{1110} \\ \texttt{1101} \\ \texttt{1011} \\ \texttt{0111} \end{array} \right.$$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not 0^n or 1^n
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2$$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

$n = 4$

$$2^n - 2 \left\{ \begin{array}{l} \texttt{1000} \\ \texttt{0100} \\ \texttt{0010} \\ \texttt{0001} \\ \texttt{1100} \\ \texttt{1010} \\ \texttt{1001} \\ \texttt{0110} \\ \texttt{0101} \\ \texttt{0011} \\ \texttt{1110} \\ \texttt{1101} \\ \texttt{1011} \\ \texttt{0111} \end{array} \right.$$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not $\texttt{0}^n$ or $\texttt{1}^n$
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1} \binom{n}{k}$$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample …

- uniformly from $\{0, \ldots, O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

$$n = 4$$

$$2^n - 2 \left\{ \begin{array}{l} \texttt{1000} \\ \texttt{0100} \\ \texttt{0010} \\ \texttt{0001} \\ \texttt{1100} \\ \texttt{1010} \\ \texttt{1001} \\ \texttt{0110} \\ \texttt{0101} \\ \texttt{0011} \\ \texttt{1110} \\ \texttt{1101} \\ \texttt{1011} \\ \texttt{0111} \end{array} \right.$$

Excursion: Uniform Non-Identical Bit Strings

`[For educational purposes only!]`

- **Goal**: Choose uniformly at random from the length n bit-strings that are not $\mathtt{0}^n$ or $\mathtt{1}^n$
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1} \binom{n}{k}$$

choose k ↲ ↳ choose k 1s in n bits

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample . . .
- uniformly from $\{0, ..., O(n + m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

$n = 4$

$$2^n - 2 \left\{ \begin{array}{l} \left.\begin{array}{l} \mathtt{1000} \\ \mathtt{0100} \\ \mathtt{0010} \\ \mathtt{0001} \end{array}\right\} k = 1 \\ \left.\begin{array}{l} \mathtt{1100} \\ \mathtt{1010} \\ \mathtt{1001} \\ \mathtt{0110} \\ \mathtt{0101} \\ \mathtt{0011} \end{array}\right\} k = 2 \\ \left.\begin{array}{l} \mathtt{1110} \\ \mathtt{1101} \\ \mathtt{1011} \\ \mathtt{0111} \end{array}\right\} k = 3 \end{array}\right.$$

Excursion: Uniform Non-Identical Bit Strings

`[For educational purposes only!]`

- **Goal**: Choose uniformly at random from the length n bit-strings that are not $\mathtt{0}^n$ or $\mathtt{1}^n$
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1} \binom{n}{k}$$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample . . .
- uniformly from $\{0, \ldots, O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

- 2-step process: choose k & choose k $\mathtt{1}$s in n bits

$n = 4$

$$2^n - 2 \left\{ \begin{array}{l} \left.\begin{array}{l} \mathtt{1000} \\ \mathtt{0100} \\ \mathtt{0010} \\ \mathtt{0001} \end{array}\right\} k = 1 \\ \left.\begin{array}{l} \mathtt{1100} \\ \mathtt{1010} \\ \mathtt{1001} \\ \mathtt{0110} \\ \mathtt{0101} \\ \mathtt{0011} \end{array}\right\} k = 2 \\ \left.\begin{array}{l} \mathtt{1110} \\ \mathtt{1101} \\ \mathtt{1011} \\ \mathtt{0111} \end{array}\right\} k = 3 \end{array} \right.$$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not 0^n or 1^n
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1} \binom{n}{k}$$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample . . .
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

- 2-step process: choose k & choose k 1s in n bits

unibs(n)

$b := 00\ldots0$ // n zeros
$k :=$ **rand**($\{1, \ldots, n-1\}$) // number of 1s
$P :=$ **randSet**($\{1, \ldots, n\}, k$) // positions of 1s
$b[P] = 1$ // set 1s in b
return b

$n = 4$

$$2^n - 2 \left\{ \begin{array}{l} \left.\begin{array}{l} 1000 \\ 0100 \\ 0010 \\ 0001 \end{array}\right\} k = 1 \\ \left.\begin{array}{l} 1100 \\ 1010 \\ 1001 \\ 0110 \\ 0101 \\ 0011 \end{array}\right\} k = 2 \\ \left.\begin{array}{l} 1110 \\ 1101 \\ 1011 \\ 0111 \end{array}\right\} k = 3 \end{array}\right.$$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not 0^n or 1^n
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n}\binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1}\binom{n}{k}$$

$$2^n = \sum_{k=0}^{n}\binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample . . .

- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

- 2-step process: choose k & choose k 1s in n bits

```
unibs(n)
  b := 00...0 // n zeros
  k := rand({1, ..., n − 1}) // number of 1s
  P := randSet({1, ..., n}, k) // positions of 1s
  b[P] = 1 // set 1s in b
  return b
```

How to sample k?

- uniform?

$$\Pr[1000] = 1/3 \cdot 1/4 = 1/12$$

$$\Pr[1100] = 1/3 \cdot 1/6 = 1/18$$

$n = 4$, $2^n - 2$:

1000, 0100, 0010, 0001	$k = 1$
1100, 1010, 1001, 0110, 0101, 0011	$k = 2$
1110, 1101, 1011, 0111	$k = 3$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not 0^n or 1^n
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1} \binom{n}{k}$$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

- 2-step process: choose k & choose k 1s in n bits

unibs(n)

$b := 00...0$ // n zeros
$k :=$ **rand**($\{1, ..., n-1\}$) // number of 1s
$P :=$ **randSet**($\{1, ..., n\}, k$) // positions of 1s
$b[P] = 1$ // set 1s in b
return b

How to sample k?
- uniform?

$$\left.\begin{aligned} \Pr[1000] &= 1/3 \cdot 1/4 = 1/12 \\ \Pr[1100] &= 1/3 \cdot 1/6 = 1/18 \end{aligned}\right\} \neq 1/14$$

$n = 4$

$$\underbrace{\left.\begin{array}{c}1000\\0100\\0010\\0001\end{array}\right\} k=1 \quad \left.\begin{array}{c}1100\\1010\\1001\\0110\\0101\\0011\end{array}\right\} k=2 \quad \left.\begin{array}{c}1110\\1101\\1011\\0111\end{array}\right\} k=3}_{2^n - 2}$$

Excursion: Uniform Non-Identical Bit Strings

`[For educational purposes only!]`

- **Goal**: Choose uniformly at random from the length n bit-strings that are not 0^n or 1^n
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n}\binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1}\binom{n}{k}$$

$$2^n = \sum_{k=0}^{n}\binom{n}{k} \qquad \binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample …
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

- 2-step process: choose k & choose k 1s in n bits

```
unibs(n)
  b := 00...0                   // n zeros
  k := rand({1, ..., n − 1})    // number of 1s
  P := randSet({1, ..., n}, k)  // positions of 1s
  b[P] = 1                      // set 1s in b
  return b
```

How to sample k?
- uniform?

$$\left.\begin{aligned}\Pr[1000] &= 1/3 \cdot 1/4 = 1/12\\ \Pr[1100] &= 1/3 \cdot 1/6 = 1/18\end{aligned}\right\} \neq 1/\underbrace{14}_{2^n-2}$$

- choose k with prob $\binom{n}{k}/(2^n - 2)$

$n = 4$

1000, 0100, 0010, 0001	$k = 1$
1100, 1010, 1001, 0110, 0101, 0011	$k = 2$
1110, 1101, 1011, 0111	$k = 3$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not 0^n or 1^n
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1} \binom{n}{k}$$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

- 2-step process: choose k & choose k 1s in n bits

unibs(n)

$b := 00...0$ // n zeros
$k :=$ **rand**$(\{1, ..., n-1\})$ // number of 1s
$P :=$ **randSet**$(\{1, ..., n\}, k)$ // positions of 1s
$b[P] = 1$ // set 1s in b
return b

How to sample k?
- uniform?

$$\left.\begin{aligned} \Pr[1000] &= 1/3 \cdot 1/4 = 1/12 \\ \Pr[1100] &= 1/3 \cdot 1/6 = 1/18 \end{aligned}\right\} \neq 1/\underbrace{14}_{2^n-2}$$

- choose k with prob $\binom{n}{k}/(2^n - 2)$
- Reduce to uniform using *Inverse Transform Sampling*

$n = 4$

$$\left.\begin{matrix}1000\\0100\\0010\\0001\end{matrix}\right\} k = 1 \qquad \left.\begin{matrix}1100\\1010\\1001\\0110\\0101\\0011\end{matrix}\right\} k = 2 \qquad \left.\begin{matrix}1110\\1101\\1011\\0111\end{matrix}\right\} k = 3$$

Excursion: Uniform Non-Identical Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit-strings that are not 0^n or 1^n
- Number of valid bit-strings:

$$2^n - 2 = \left(\sum_{k=0}^{n} \binom{n}{k}\right) - 2 = \sum_{k=1}^{n-1} \binom{n}{k}$$

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Assumptions: We can sample ...

- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

- 2-step process: choose k & choose k 1s in n bits

unibs(n)

$b := 00\ldots0$ // n zeros
$k :=$ **rand**$(\{1, \ldots, n-1\})$ // number of 1s
$P :=$ **randSet**$(\{1, \ldots, n\}, k)$ // positions of 1s
$b[P] = 1$ // set 1s in b
return b

How to sample k?

- uniform?

$$\left.\begin{aligned} \Pr[1000] &= 1/3 \cdot 1/4 = 1/12 \\ \Pr[1100] &= 1/3 \cdot 1/6 = 1/18 \end{aligned}\right\} \neq 1/\underbrace{14}_{2^n-2}$$

- choose k with prob $\binom{n}{k}/(2^n - 2)$
- Reduce to uniform using *Inverse Transform Sampling*

How to sample P?

$n = 4$

$$\left.\begin{matrix} 1000 \\ 0100 \\ 0010 \\ 0001 \end{matrix}\right\} k = 1 \qquad \left.\begin{matrix} 1100 \\ 1010 \\ 1001 \\ 0110 \\ 0101 \\ 0011 \end{matrix}\right\} k = 2 \qquad \left.\begin{matrix} 1110 \\ 1101 \\ 1011 \\ 0111 \end{matrix}\right\} k = 3$$

Excursion-Excursion: Reservoir Sampling

`[For educational purposes only!]`

- **Goal**: Choose a set of size k uniformly at random from the n elements.

Assumptions: We can sample . . .

- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample . . .
- uniformly from $\{0, ..., O(n + m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

```
randSet({1, ..., n}, k)
  r := [1, ..., k] // reservoir
  for i from k + 1 to n do
    j := unif({1, ..., i})
    if j ≤ k then r[j] = i
  return r
```

Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet($\{1, \dots, n\}, k$)
- $r := [1, \dots, k]$ // reservoir
- **for** i from $k+1$ to n **do**
  - $j :=$ **unif**($\{1, \dots, i\}$)
  - **if** $j \leq k$ **then** $r[j] = i$
- **return** r

1	2	3	4	5	6	7	8

1	2	3

Excursion-Excursion: Reservoir Sampling

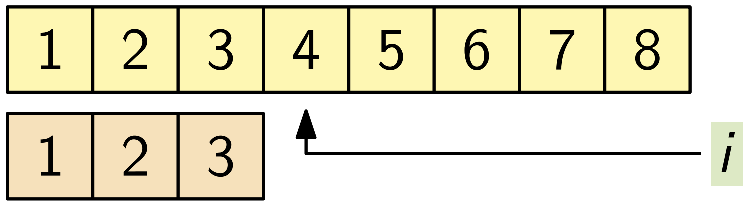
[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0,1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet$(\{1, \ldots, n\}, k)$
- $r := [1, \ldots, k]$ // reservoir
- **for** i from $k+1$ to n **do**
  - $j :=$ **unif**$(\{1, \ldots, i\})$
  - **if** $j \le k$ **then** $r[j] = i$
- **return** r



Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

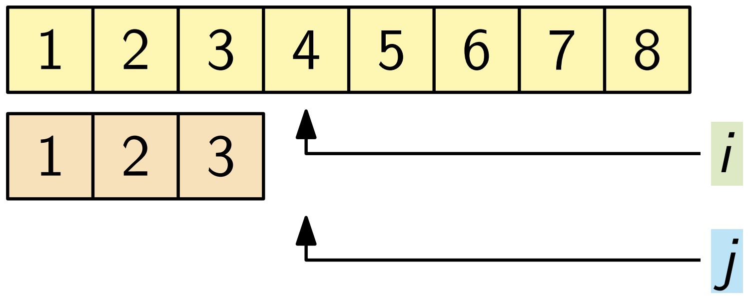
- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet$(\{1, \ldots, n\}, k)$

$r := [1, \ldots, k]$ // reservoir
for i from $k+1$ to n **do**
  $j :=$ **unif**$(\{1, \ldots, i\})$
  if $j \le k$ **then** $r[j] = i$
return r



Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample . . .
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet$(\{1, \ldots, n\}, k)$

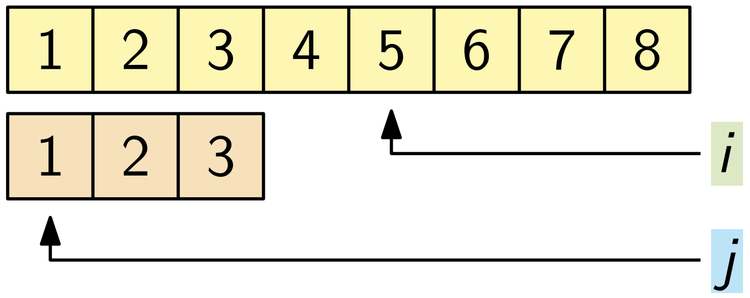
$r := [1, \ldots, k]$ // reservoir

for i from $k+1$ to n **do**

  $j :=$ **unif**$(\{1, \ldots, i\})$

  if $j \leq k$ **then** $r[j] = i$

return r



Excursion-Excursion: Reservoir Sampling

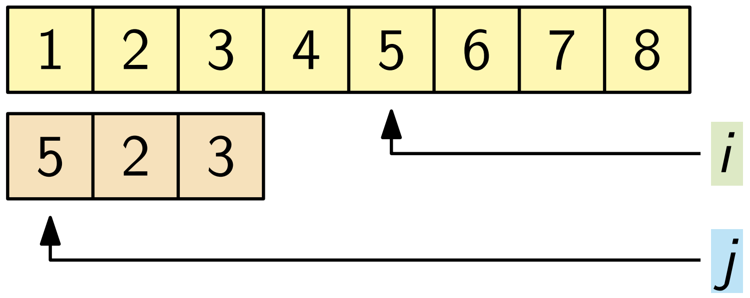
[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0,1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet$(\{1, \ldots, n\}, k)$
- $r := [1, \ldots, k]$ // reservoir
- **for** i from $k+1$ to n **do**
  - $j :=$ **unif**$(\{1, \ldots, i\})$
  - **if** $j \leq k$ **then** $r[j] = i$
- **return** r



Excursion-Excursion: Reservoir Sampling

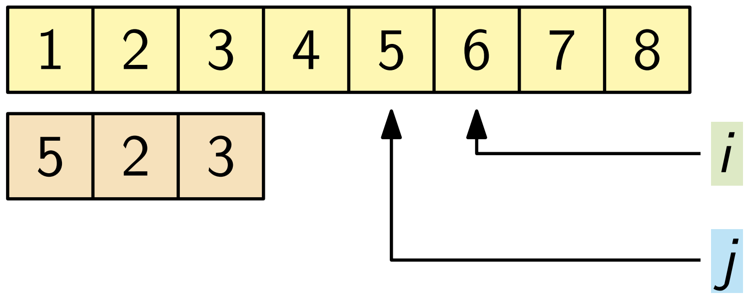
[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet$(\{1, \ldots, n\}, k)$
- $r := [1, \ldots, k]$ // reservoir
- **for** i from $k+1$ to n **do**
  - $j :=$ **unif**$(\{1, \ldots, i\})$
  - **if** $j \leq k$ **then** $r[j] = i$
- **return** r



Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet$(\{1, \ldots, n\}, k)$

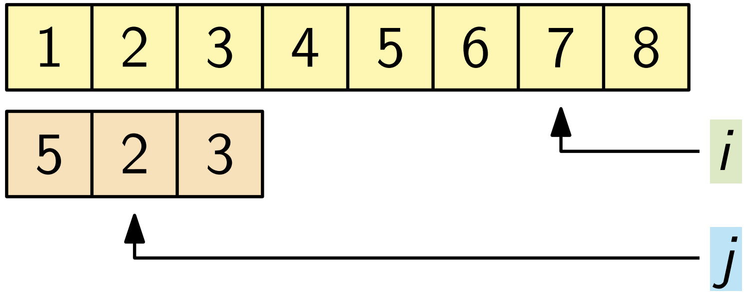
$r := [1, \ldots, k]$ // reservoir

for i from $k+1$ to n **do**

  $j :=$ **unif**$(\{1, \ldots, i\})$

  if $j \leq k$ **then** $r[j] = i$

return r



Excursion-Excursion: Reservoir Sampling

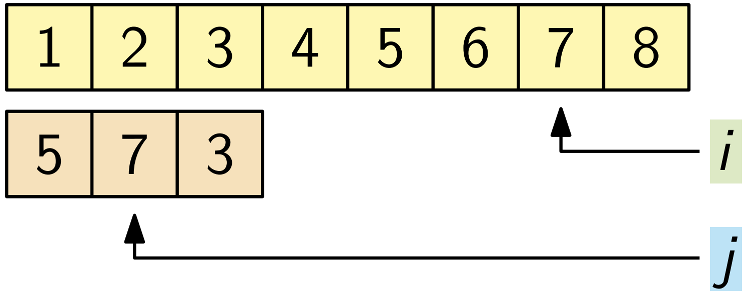
[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample …
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet($\{1, \ldots, n\}, k$)
- $r := [1, \ldots, k]$ // reservoir
- **for** i from $k+1$ to n **do**
  - $j :=$ **unif**($\{1, \ldots, i\}$)
  - **if** $j \leq k$ **then** $r[j] = i$
- **return** r



Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

> **Assumptions**: We can sample ...
> - uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
> - uniformly from $[0, 1]$ in $O(1)$ time
>
> Not possible in theory. Reasonable in practice.

randSet$(\{1, \dots, n\}, k)$

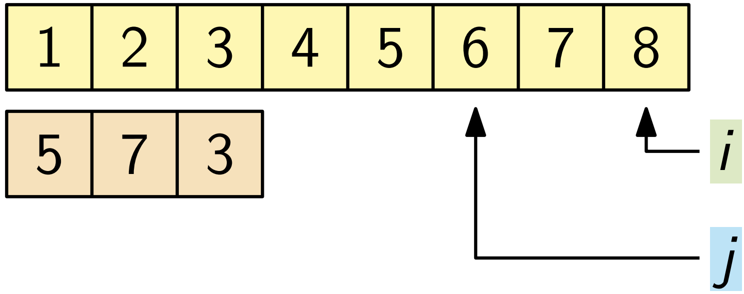
$r := [1, \dots, k]$ // reservoir

for i from $k+1$ to n **do**

  $j :=$ **unif**$(\{1, \dots, i\})$

  if $j \le k$ **then** $r[j] = i$

return r



Excursion-Excursion: Reservoir Sampling

[For educational purposes only!]

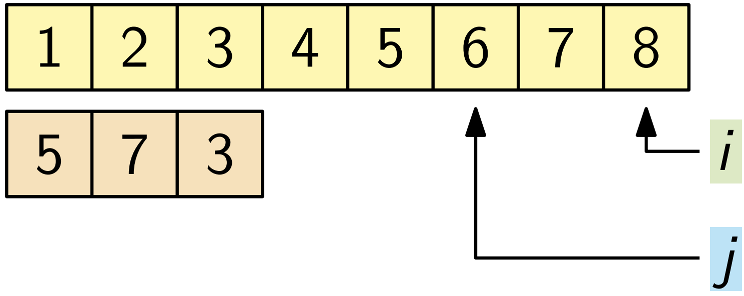
- **Goal**: Choose a set of size k uniformly at random from the n elements.
- **Idea**:
  - initialize **reservoir** with first k elements
  - replace reservoir elements at random

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0,1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

randSet$(\{1, \ldots, n\}, k)$

$r := [1, \ldots, k]$ // reservoir // $O(k)$
for i from $k+1$ to n **do** // $O(n-k)$
  $j :=$ **unif**$(\{1, \ldots, i\})$ // $O(1)$
  if $j \leq k$ **then** $r[j] = i$ // $O(1)$
return r



- **Running time**: $O(n)$

Excursion: Uniform Non-Homogeneous Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit strings that are not 0^n or 1^n
- 2-step process:
  - choose k
  - choose k 1s in n bits

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

unibs(n)

```
b := 00...0 // n zeros
k := rand({1, ..., n − 1}) // number of 1s
P := randSet({1, ..., n}, k) // positions of 1s
b[P] = 1 // set 1s in b
return b
```

Excursion: Uniform Non-Homogeneous Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit strings that are not 0^n or 1^n
- 2-step process:
  - choose k
  - choose k 1s in n bits

Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0,1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

unibs(n)

$b := 00\ldots0$ // n zeros // $O(n)$

$k :=$ **rand**$(\{1, \ldots, n-1\})$ // number of 1s // $O(\log(n))$ via Inverse Transform Sampling

$P :=$ **randSet**$(\{1, \ldots, n\}, k)$ // positions of 1s // $O(n)$ via Reservoir Sampling

$b[P] = 1$ // set 1s in b // $O(k) \subseteq O(n)$

return b

Excursion: Uniform Non-Homogeneous Bit Strings

[For educational purposes only!]

- **Goal**: Choose uniformly at random from the length n bit strings that are not 0^n or 1^n
- 2-step process:
  - choose k
  - choose k 1s in n bits

Assumptions: We can sample . . .
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

unibs(n)

```
b := 00...0 // n zeros                                  // O(n)
k := rand({1, ..., n − 1}) // number of 1s              // O(log(n)) via Inverse Transform Sampling
P := randSet({1, ..., n}, k) // positions of 1s         // O(n) via Reservoir Sampling
b[P] = 1 // set 1s in b                                 // O(k) ⊆ O(n)
return b
```

Under our assumptions, we can sample a length n bit string that is not 0^n or 1^n uniformly at random in time $O(n)$.

Simple Randomized Cut

Karlsruhe Institute of Technology

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Simple Randomized Cut

Karlsruhe Institute of Technology

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Amplification

- Repeat the algorithm to obtain t independent random cuts, return the smallest

  $\Pr[\text{“minimum found”}] \geq 1 - \left(1 - 1/(2^{n-1} - 1)\right)^t$

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Amplification

- Repeat the algorithm to obtain t independent random cuts, return the smallest

$$\Pr[\text{“minimum found”}] \geq 1 - \left(1 - \underbrace{1/(2^{n-1} - 1)}_{\text{minimum}}\right)^t$$

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

> **Observation**: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Amplification

- Repeat the algorithm to obtain t independent random cuts, return the smallest

$$\Pr[\text{“minimum found”}] \geq 1 - \Big(\underbrace{1 -}_{\text{not}} \underbrace{1/(2^{n-1} - 1)}_{\text{minimum}}\Big)^t$$

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Amplification

- Repeat the algorithm to obtain t independent random cuts, return the smallest

$$\Pr[\text{“minimum found”}] \geq 1 - \big(\underbrace{1 -}_{\text{not}} \underbrace{1/(2^{n-1} - 1)}_{\text{minimum}}\big)^{\overset{t}{\downarrow\, t \text{ times}}}$$

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Amplification

- Repeat the algorithm to obtain t independent random cuts, return the smallest

  $\Pr[\text{“minimum found”}] \geq 1 - \left(1 - 1/(2^{n-1} - 1)\right)^t \geq 1 - e^{-t/(2^{n-1}-1)}$

$1 + x \leq e^x$ for $x \in \mathbb{R}$

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Amplification

- Repeat the algorithm to obtain t independent random cuts, return the smallest

  $\Pr[\text{“minimum found”}] \geq 1 - \left(1 - 1/(2^{n-1} - 1)\right)^t \geq 1 - e^{-t/(2^{n-1}-1)}$

  $1 + x \leq e^x$ for $x \in \mathbb{R}$

- For $t = 2^{n-1} - 1$ minimum found with constant probability $1 - 1/e \approx 0.63$

Simple Randomized Cut

- Simple idea: choose a cut uniformly at random among all possible cuts and return it.
- Running time: $O(n)$ much better than the $\Omega(n^3)$ in the deterministic setting , but...

Success probability

- $2^{n-1} - 1$ cuts in a graph with n nodes
- How many min-cuts? $\rightarrow$ pessimistic assumption: 1

Observation: On a graph with n nodes, **Simple Randomized Cut** runs in $O(n)$ time and returns a minimum cut with probability at least $1/(2^{n-1} - 1)$.$\rightarrow$ exponentially small!

Amplification

- Repeat the algorithm to obtain t independent random cuts, return the smallest

  $\Pr[\text{“minimum found”}] \geq 1 - \left(1 - 1/(2^{n-1} - 1)\right)^t \geq 1 - e^{-t/(2^{n-1}-1)}$

  $1 + x \leq e^x$ for $x \in \mathbb{R}$

- For $t = 2^{n-1} - 1$ minimum found with constant probability $1 - 1/e \approx 0.63$
- For $t = (2^{n-1} - 1) \cdot \log(n)$ minimum found with high probability $1 - 1/n$

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*

		Correct Answer	
		✗	✓
Algo Output	✗	true neg	false neg
	✓	false pos	true pos

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*

✗ answers are always correct
✓ answers are correct with bounded probability

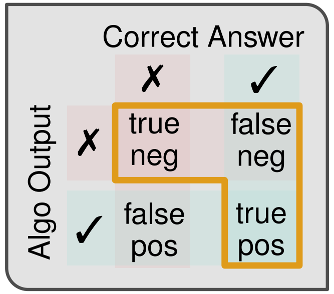
Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*

✓ answers are always correct
✗ answers are correct with bounded probability



Probability Amplification

Karlsruhe Institute of Technology

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*
  - **Two-sided error**: $\boxed{\textit{no bias}}$

✗ answers are correct with bounded probability
✓ answers are correct with bounded probability

Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*
  - **Two-sided error**: *no bias*
- In optimization problems p is the probability of finding the optimum

Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*
  - **Two-sided error**: *no bias*
- In optimization problems p is the probability of finding the optimum

Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Definition: **Probability amplification** is the process of increasing the success probability of a Monte Carlo algorithm by using multiple runs.

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*
  - **Two-sided error**: *no bias*
- In optimization problems p is the probability of finding the optimum

bias

majority

best

Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Definition: **Probability amplification** is the process of increasing the success probability of a Monte Carlo algorithm by using multiple runs.

- After t (independent) runs return the ...

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*
  - **Two-sided error**: *no bias*
- In optimization problems p is the probability of finding the optimum

bias

majority

best

Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Definition: **Probability amplification** is the process of increasing the success probability of a Monte Carlo algorithm by using multiple runs.

- After t (independent) runs return the ...

  $\Pr[\text{“success”}] \geq 1 - (1-p)^t \geq 1 - e^{-pt}$ (for two-sided errors it's a bit more complicated)
- Error probability decreases exponentially in t

Probability Amplification

Definition: A **Monte Carlo Algorithm** is a randomized algorithm that terminates deterministically and whose output is correct only with a certain probability $p \in (0, 1)$.

- In decision problems p is the probability of giving the correct answer
  - **One-sided error**: either *false-biased* or *true-biased*
  - **Two-sided error**: *no bias*
- In optimization problems p is the probability of finding the optimum

bias
majority
best

		Correct Answer	
		✗	✓
Algo Output	✗	true neg	false neg
	✓	false pos	true pos

Definition: **Probability amplification** is the process of increasing the success probability of a Monte Carlo algorithm by using multiple runs.

- After t (independent) runs return the . . .

  $\Pr[\text{“success”}] \geq 1 - (1 - p)^t \geq 1 - e^{-pt}$ (for two-sided errors it's a bit more complicated)

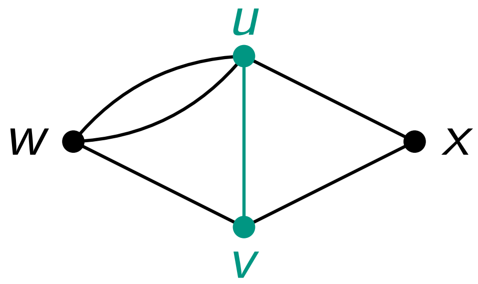
- Error probability decreases exponentially in t

For Simple Randomized Cut we had to pay with exponentially large running time . . .

Karger’s Algorithm

Edge Contraction

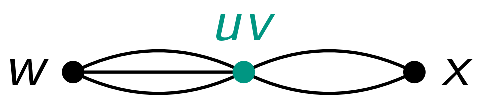
- Merge two adjacent nodes in a multigraph without self-loops



Karger's Algorithm

Edge Contraction

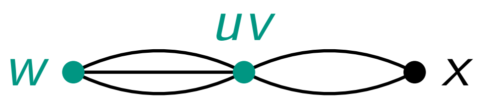
- Merge two adjacent nodes in a multigraph without self-loops



Karger’s Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops



Karger’s Algorithm

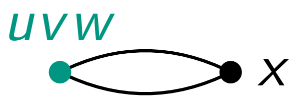
Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut
  & hope there are few essential ones

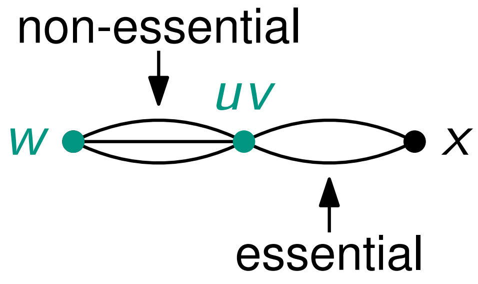
Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut

Contraction Algorithm

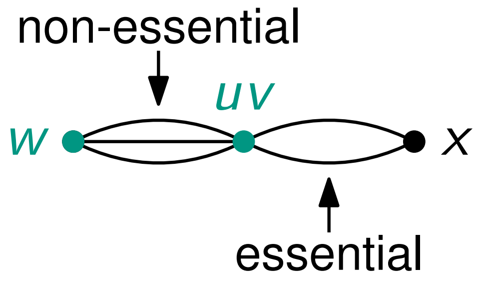
- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$
  as well as *essential* edges } part of a min-cut
  & hope there are few essential ones



Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut & hope there are few essential ones

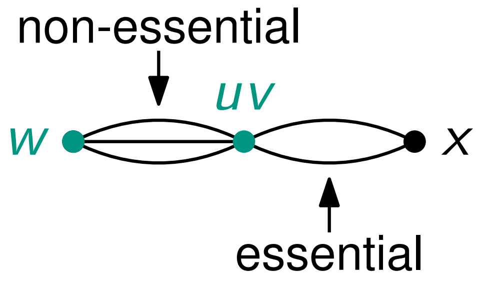
Karger$(G_0 = (V_0, E_0))$

> **for** $i = 1$ to $n - 2$ **do**
>
> > $e :=$ **unif**(E_{i-1})
> >
> > $G_i = G_{i-1}.$**contract**(e)
>
> **return** unique cut in G_{n-2}

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut
  & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

  for $i = 1$ to $n - 2$ **do** // $O(n)$

    $e :=$ **unif**(E_{i-1}) // $O(1)$

    $G_i = G_{i-1}$.**contract**(e) // $O(n)$

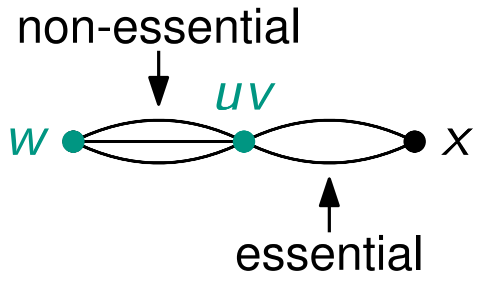
  return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges (part of a min-cut) & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

  for $i = 1$ to $n - 2$ **do** // $O(n)$
    $e :=$ **unif**(E_{i-1}) // $O(1)$
    $G_i = G_{i-1}$.**contract**(e) // $O(n)$
  return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

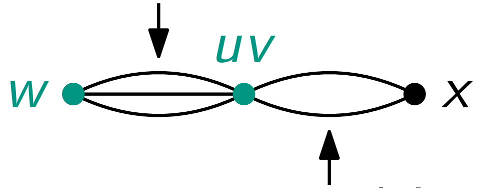
Observation: A cut in G_i is a cut in G_0.

Because endpoints of removed edges (self-loops) are within the same part in a cut in G_i.

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges $\}$ part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

```
for i = 1 to n − 2 do              // O(n)
    e := unif(E_{i−1})             // O(1)
    G_i = G_{i−1}.contract(e)      // O(n)
return unique cut in G_{n−2}
```

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

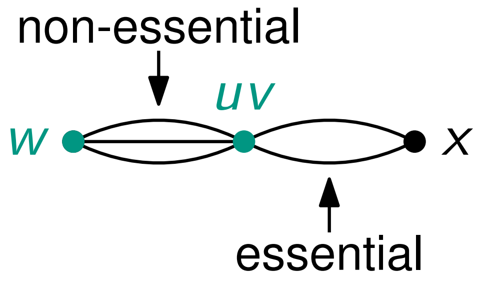
Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut
  & hope there are few essential ones

$\textbf{Karger}(G_0 = (V_0, E_0))$

for $i = 1$ to $n-2$ **do** // $O(n)$
  $e := \textbf{unif}(E_{i-1})$ // $O(1)$
  $G_i = G_{i-1}.\textbf{contract}(e)$ // $O(n)$
return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

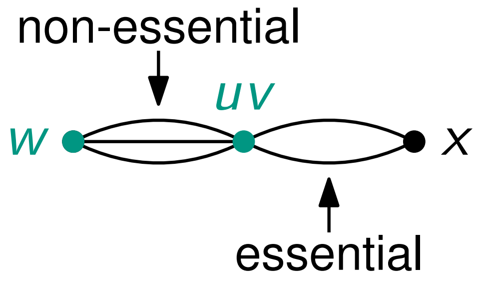
- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

$\Pr[\mathcal{E}_1] = 1 - \frac{k}{m}$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges $\}$ part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

```
for i = 1 to n − 2 do               // O(n)
    e := unif(E_{i−1})               // O(1)
    G_i = G_{i−1}.contract(e)        // O(n)
return unique cut in G_{n−2}
```

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

o.w. ↯

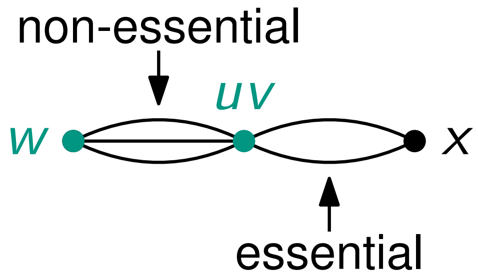
- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

$\Pr[\mathcal{E}_1] = 1 - \frac{k}{m}$

Karger’s Algorithm

Edge Contraction



- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut

Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges (part of a min-cut) & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

for $i = 1$ to $n - 2$ **do** $\quad // O(n)$

$\quad e :=$ **unif**(E_{i-1}) $\quad // O(1)$

$\quad G_i = G_{i-1}.$**contract**(e) $\quad // O(n)$

return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

o.w. ↯

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ “C in G_i”

Observation: min-degree $\geq k$

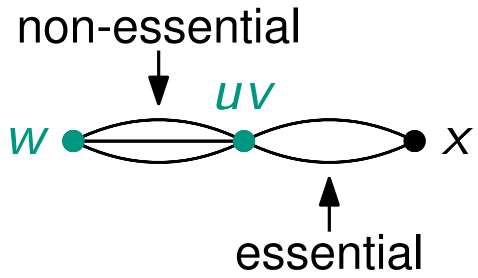
$\Pr[\mathcal{E}_1] = 1 - \frac{k}{m}$

(holds for all G_i due to 1st observation)

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges $\}$ part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

  for $i = 1$ to $n - 2$ **do** // $O(n)$

    $e :=$ **unif**(E_{i-1}) // $O(1)$

    $G_i = G_{i-1}$.**contract**(e) // $O(n)$

  return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

o.w. ↯

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

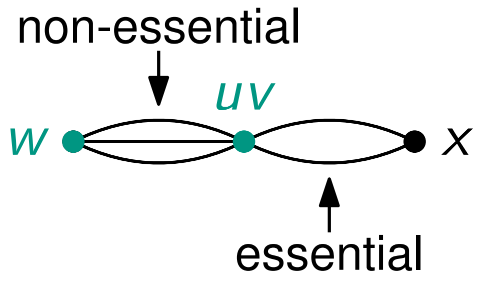
$\Pr[\mathcal{E}_1] = 1 - \frac{k}{m}$

$$m = \frac{1}{2}\sum_{v\in V}\deg(v) \geq \frac{1}{2}\sum_{v\in V}k \geq \frac{1}{2}nk$$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as $\left.\textit{essential}\text{ edges}\right\}$ part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

- **for** $i = 1$ to $n-2$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}$.**contract**(e) // $O(n)$
- **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i$ = "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

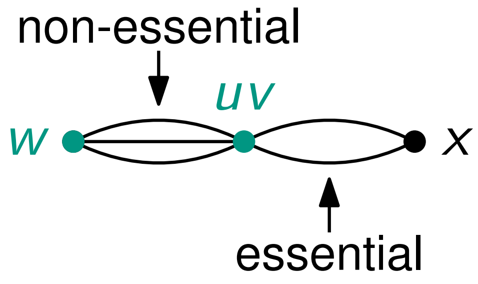
$$\Pr[\mathcal{E}_1] = 1 - \frac{k}{m} \geq 1 - \frac{k}{nk/2} = 1 - \frac{2}{n}$$

$$m = \frac{1}{2}\sum_{v \in V} \deg(v) \geq \frac{1}{2}\sum_{v \in V} k \geq \frac{1}{2}nk$$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as $\left.\textit{essential}\text{ edges}\right\}$ part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

  for $i = 1$ to $n-2$ **do** $\quad // O(n)$

    $e :=$ **unif**(E_{i-1}) $\quad // O(1)$

    $G_i = G_{i-1}$.**contract**(e) $\quad // O(n)$

  return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

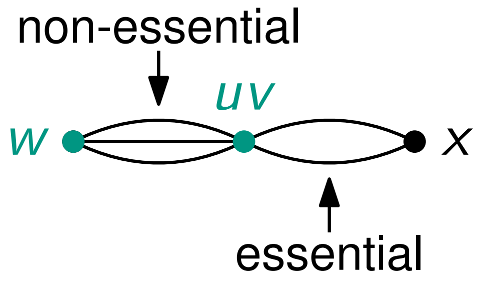
$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

Karger's Algorithm

Karlsruhe Institute of Technology

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges $\}$ part of a min-cut
  & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

- **for** $i = 1$ to $n - 2$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}$.**contract**(e) // $O(n)$
- **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1}$

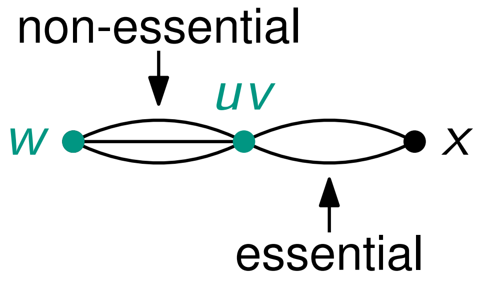
none of the k edges of C contracted

do not contract k edges in an $n - 1$-node graph

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut
  & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

- **for** $i = 1$ to $n-2$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}$.**contract**(e) // $O(n)$
- **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

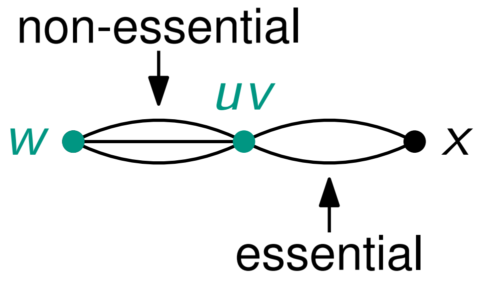
$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

  for $i = 1$ to $n - 2$ **do** // $O(n)$

    $e :=$ **unif**(E_{i-1}) // $O(1)$

    $G_i = G_{i-1}$.**contract**(e) // $O(n)$

  return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$ (holds for all G_i due to 1st observation)

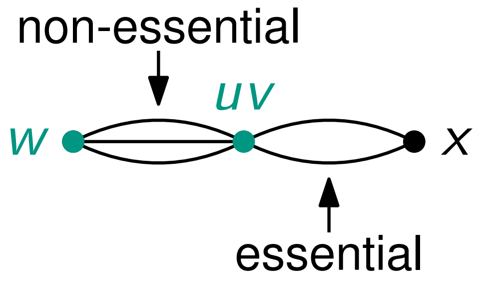
$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

$$\underbrace{\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]}_{\text{chain rule of probability}}$$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish *non-essential* (not part of a min-cut) as well as *essential* edges (part of a min-cut) & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

  for $i = 1$ to $n-2$ **do** // $O(n)$

    $e :=$ **unif**(E_{i-1}) // $O(1)$

    $G_i = G_{i-1}$.**contract**(e) // $O(n)$

  return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

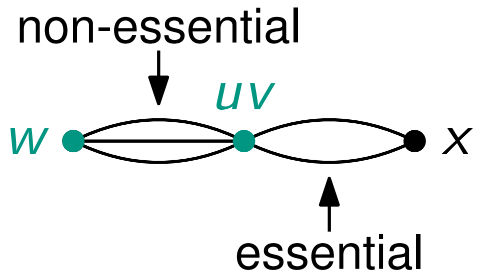
$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

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

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish *non-essential* (not part of a min-cut) as well as *essential* edges (part of a min-cut) & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

- **for** $i = 1$ to $n-2$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}.$**contract**(e) // $O(n)$
- **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

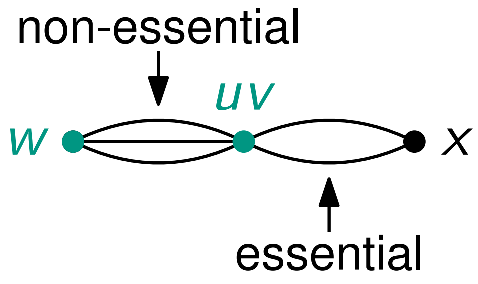
$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{4}\right)\left(1 - \frac{2}{3}\right)$$

$1 = \frac{n-i}{n-i}$

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as $\left.\textit{essential}\right\}$ edges part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

- **for** $i = 1$ to $n-2$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}.$**contract**(e) // $O(n)$
- **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

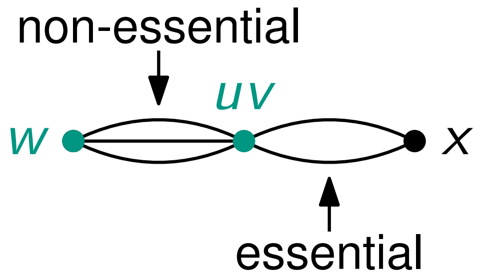
$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

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

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges $\}$ part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

- **for** $i = 1$ to $n - 2$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}$.**contract**(e) // $O(n)$
- **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$ (holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

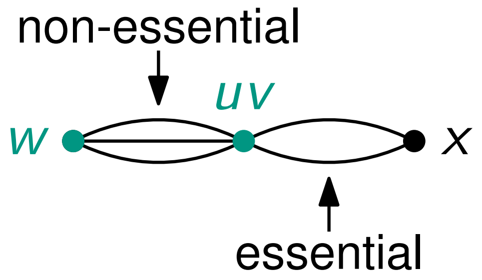
$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

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

Karger's Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as $\underbrace{\textit{essential}}_{\text{part of a min-cut}}$ edges & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

> **for** $i = 1$ to $n - 2$ **do** // $O(n)$
>
> > $e :=$ **unif**(E_{i-1}) // $O(1)$
> >
> > $G_i = G_{i-1}$.**contract**(e) // $O(n)$
>
> **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

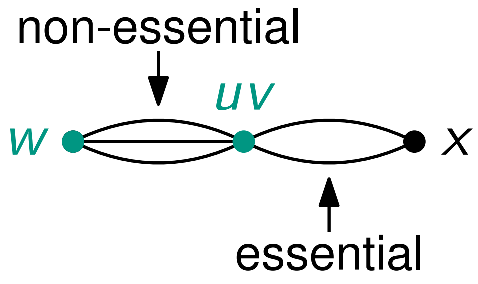
$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

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

Karger's Algorithm



Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut

Contraction Algorithm

- Motivation: distinguish *non-essential* (not part of a min-cut) as well as *essential* edges (part of a min-cut) & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

```
for i = 1 to n − 2 do            // O(n)
    e := unif(E_{i−1})           // O(1)
    G_i = G_{i−1}.contract(e)    // O(n)
return unique cut in G_{n−2}
```

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

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

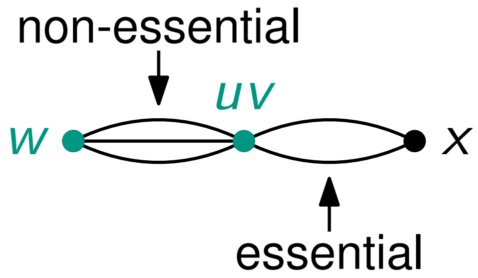
Institute of Theoretical Informatics, Algorithm Engineering & Scalable Algorithms

Karger's Algorithm

Karlsruhe Institute of Technology

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$

- **for** $i = 1$ to $n - 2$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}$.**contract**(e) // $O(n)$
- **return** unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i =$ "C in G_i"

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

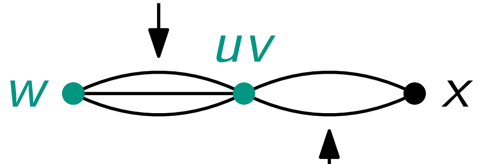
$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

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

Karger’s Algorithm

Edge Contraction

- Merge two adjacent nodes in a multigraph without self-loops
- A (multi) graph with two nodes has a unique cut



Contraction Algorithm

- Motivation: distinguish $\overbrace{\textit{non-essential}}^{\text{not part of a min-cut}}$ as well as *essential* edges } part of a min-cut
  & hope there are few essential ones

Karger$(G_0 = (V_0, E_0))$
  for $i = 1$ to $n - 2$ **do** // $O(n)$
    $e :=$ **unif**(E_{i-1}) // $O(1)$
    $G_i = G_{i-1}$.**contract**(e) // $O(n)$
  return unique cut in G_{n-2}

- Running time in $O(n^2)$
- Can be implemented to run in $O(m)$

Success Probability

Observation: A cut in G_i is a cut in G_0.

- Consider min-cut with cut set C and $|C| = k$
- $\mathcal{E}_i$ = “C in G_i”

Observation: min-degree $\geq k$

(holds for all G_i due to 1st observation)

$\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$

$\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - \frac{2}{n-1} \longrightarrow \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{n-i+1}$

$$\Pr[\mathcal{E}_{n-2}] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-3}]$$

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

$$\geq \frac{2}{n\,(n-1)}$$

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

Success probability $\geq p$

Number of repetitions t

Amplified prob. $\geq 1 - e^{-pt}$

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2} \log(n)$

Success probability $\geq p$

Number of repetitions t

Amplified prob. $\geq 1 - e^{-pt}$

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2}\log(n)$

Success probability $\geq p$

Number of repetitions t

Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2 \log(n))$ Karger repetitions run in $O(n^4 \log(n))$ total time and return a min-cut with high probability.

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2}\log(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2\log(n))$ Karger repetitions run in $O(n^4\log(n))$ total time and return a min-cut with high probability.

Much better than exp. time Simple Randomized Cut!

Karger's Algorithm Amplified

Karlsruhe Institute of Technology

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2}\log(n)$

Success probability $\geq p$

Number of repetitions t

Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2 \log(n))$ Karger repetitions run in $O(n^4 \log(n))$ total time and return a min-cut with high probability. Much better than exp. time Simple Randomized Cut!

Sidenote: Number of minimum cuts

- Let $C_1, \ldots, C_\ell$ be all the min-cuts in G and $\mathcal{E}^i_{n-2}$ for $i \in [\ell]$ be the event that C_i is returned by Karger's algorithm

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2}\log(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2 \log(n))$ Karger repetitions run in $O(n^4 \log(n))$ total time and return a min-cut with high probability.

Much better than exp. time Simple Randomized Cut!

Sidenote: Number of minimum cuts

- Let $C_1, \ldots, C_\ell$ be all the min-cuts in G and $\mathcal{E}^i_{n-2}$ for $i \in [\ell]$ be the event that C_i is returned by Karger's algorithm
- Just seen: $\Pr[\mathcal{E}^i_{n-2}] \geq \frac{2}{n(n-1)}$

Institute of Theoretical Informatics, Algorithm Engineering & Scalable Algorithms

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{"min-cut found"}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2}\log(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2 \log(n))$ Karger repetitions run in $O(n^4 \log(n))$ total time and return a min-cut with high probability. Much better than exp. time Simple Randomized Cut!

Sidenote: Number of minimum cuts

- Let $C_1, \ldots, C_\ell$ be all the min-cuts in G and $\underbrace{\mathcal{E}^i_{n-2}}_{\text{disjoint, since the algorithm returns only one cut}}$ for $i \in [\ell]$ be the event that C_i is returned by Karger's algorithm
- Just seen: $\Pr[\mathcal{E}^i_{n-2}] \geq \frac{2}{n(n-1)}$

Karger's Algorithm Amplified

Karlsruhe Institute of Technology

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2}\log(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2 \log(n))$ Karger repetitions run in $O(n^4 \log(n))$ total time and return a min-cut with high probability. Much better than exp. time Simple Randomized Cut!

Sidenote: Number of minimum cuts

- Let $C_1, \ldots, C_\ell$ be all the min-cuts in G and $\underbrace{\mathcal{E}^i_{n-2}}_{\text{disjoint, since the algorithm returns only one cut}}$ for $i \in [\ell]$ be the event that C_i is returned by Karger's algorithm
- Just seen: $\Pr[\mathcal{E}^i_{n-2}] \geq \frac{2}{n(n-1)}$

$$\Pr\left[\bigcup_{i\in[\ell]} \mathcal{E}^i_{n-2}\right] = \sum_{i\in[\ell]} \Pr[\mathcal{E}^i_{n-2}] \geq \frac{2\cdot\ell}{n(n-1)}$$

Karger's Algorithm Amplified

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2} \log(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2 \log(n))$ Karger repetitions run in $O(n^4 \log(n))$ total time and return a min-cut with high probability.

Much better than exp. time Simple Randomized Cut!

Sidenote: Number of minimum cuts

- Let $C_1, \ldots, C_\ell$ be all the min-cuts in G and $\underbrace{\mathcal{E}^i_{n-2}}_{\text{disjoint, since the algorithm returns only one cut}}$ for $i \in [\ell]$ be the event that C_i is returned by Karger's algorithm
- Just seen: $\Pr[\mathcal{E}^i_{n-2}] \geq \frac{2}{n(n-1)}$

$$1 \geq \Pr\left[\bigcup_{i\in[\ell]} \mathcal{E}^i_{n-2}\right] = \sum_{i\in[\ell]} \Pr[\mathcal{E}^i_{n-2}] \geq \frac{2\cdot\ell}{n(n-1)}$$

Karger's Algorithm Amplified

Karlsruhe Institute of Technology

Theorem: On a graph with n nodes, Karger's algorithm runs in $O(n^2)$ time and returns a minimum cut with probability at least $2/(n(n-1))$.

$$\Pr[\text{"min-cut found"}] \geq 1 - \exp\left(-\frac{2t}{n(n-1)}\right) = 1 - \frac{1}{n}$$

for $t = \frac{n(n-1)}{2}\log(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(n^2 \log(n))$ Karger repetitions run in $O(n^4 \log(n))$ total time and return a min-cut with high probability.

Much better than exp. time Simple Randomized Cut!

Sidenote: Number of minimum cuts

- Let $C_1, \ldots, C_\ell$ be all the min-cuts in G and $\underbrace{\mathcal{E}^i_{n-2}}_{\text{disjoint, since the algorithm returns only one cut}}$ for $i \in [\ell]$ be the event that C_i is returned by Karger's algorithm
- Just seen: $\Pr[\mathcal{E}^i_{n-2}] \geq \frac{2}{n(n-1)}$

$$1 \geq \Pr\left[\bigcup_{i\in[\ell]} \mathcal{E}^i_{n-2}\right] = \sum_{i\in[\ell]} \Pr[\mathcal{E}^i_{n-2}] \geq \frac{2\cdot\ell}{n(n-1)}$$

Observation: A graph on n nodes contains at most $\frac{n(n-1)}{2}$ minimum cuts.

More Amplification: Karger-Stein

Karlsruhe Institute of Technology

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

More Amplification: Karger-Stein

Karlsruhe Institute of Technology

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right) \cdots \left(\frac{n-i+2-2}{n-i+2}\right)\left(\frac{n-i+1-2}{n-i+1}\right)$$

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{n-i}{n-i+2}\right)\left(\frac{n-i-1}{n-i+1}\right)$$

More Amplification: Karger-Stein

Karlsruhe Institute of Technology

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right) \cdots \left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

More Amplification: Karger-Stein

Karlsruhe Institute of Technology

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\cdots\left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives *i* contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\cdots\left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse

More Amplification: Karger-Stein

Karlsruhe Institute of Technology

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right) \cdots \left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(n - (n - n/\sqrt{2} - 1))(n - (n - n/\sqrt{2} - 1) - 1)}{n(n-1)}$$

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\begin{aligned}
\Pr[\mathcal{E}_i] &= \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \\
&\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right) \\
&= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\cdots\left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right) \\
&= \frac{(n - i)(n - i - 1)}{n(n-1)}
\end{aligned}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(n - \quad n + n/\sqrt{2} + 1\ )(n - \quad n + n/\sqrt{2} + 1\ \ - 1)}{n(n-1)}$$

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{n-i+2}\right)\left(1-\frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\cdots\left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(\cancel{n-n} + n/\sqrt{2}+1\ )(\cancel{n-n} + n/\sqrt{2} + \cancel{1-1})}{n(n-1)}$$

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right) \cdots \left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(\cancel{n - n} + n/\sqrt{2} + 1\ )(\cancel{n - n} + n/\sqrt{2} + \cancel{1 - 1})}{n(n-1)} = \frac{n^2/2 + n/\sqrt{2}}{n(n-1)}$$

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right) \cdots \left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(\cancel{n-n} + n/\sqrt{2} + 1\ )(\cancel{n-n} + n/\sqrt{2} + \cancel{1-1})}{n(n-1)} = \frac{n^2/2 + n/\sqrt{2}}{n(n-1)} = \frac{n(n/2 + 1/\sqrt{2})}{n(n-1)}$$

More Amplification: Karger-Stein

Karlsruhe Institute of Technology

Motivation

- Probability that a min-cut survives i contractions

$$\begin{aligned}
\Pr[\mathcal{E}_i] &= \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \\
&\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right) \\
&= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\cdots\left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right) \\
&= \frac{(n-i)(n-i-1)}{n(n-1)}
\end{aligned}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(\cancel{n - n} + n/\sqrt{2} + 1)(\cancel{n - n} + n/\sqrt{2} + \cancel{1 - 1})}{n(n-1)} = \frac{n^2/2 + n/\sqrt{2}}{n(n-1)} = \frac{\cancel{n}(n/2 + 1/\sqrt{2})}{\cancel{n}(n-1)}$$

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{n-i+2}\right)\left(1-\frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\cdots\left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(\cancel{n-n}+n/\sqrt{2}+1\ )(\cancel{n-n}+n/\sqrt{2}+\cancel{1-1})}{n(n-1)} = \frac{n^2/2+n/\sqrt{2}}{n(n-1)} = \frac{\cancel{n}(n/2+1/\sqrt{2})}{\cancel{n}(n-1)} = \frac{1}{2}\cdot\frac{n+\sqrt{2}}{n-1}$$

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\cdots\left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases
- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(\cancel{n - n} + n/\sqrt{2} + 1)(\cancel{n - n} + n/\sqrt{2} + \cancel{1 - 1})}{n(n-1)} = \frac{n^2/2 + n/\sqrt{2}}{n(n-1)} = \frac{\cancel{n}(n/2 + 1/\sqrt{2})}{\cancel{n}(n-1)} = \frac{1}{2} \cdot \underbrace{\frac{n+\sqrt{2}}{n-1}}_{\geq 1} \geq \frac{1}{2}$$

- Probability that no mistake made after t steps still large

More Amplification: Karger-Stein

Motivation

- Probability that a min-cut survives i contractions

$$\Pr[\mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_i \mid \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-i+2}\right)\left(1 - \frac{2}{n-i+1}\right)$$

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right) \cdots \left(\frac{n-i}{\cancel{n-i+2}}\right)\left(\frac{n-i-1}{\cancel{n-i+1}}\right)$$

$$= \frac{(n-i)(n-i-1)}{n(n-1)}$$

- With increasing number of steps the probability for a min-cut to survive decreases

KargerStein$(G_0 = (V_0, E_0))$
- **if** $|V_0| = 2$ **then** return unique cut
- **for** $i = 1$ to $t = |V_0| - \frac{|V_0|}{\sqrt{2}} - 1$ **do**
  - $e :=$ **unif**(E_{i-1})
  - $G_i = G_{i-1}$.**contract**(e)
- $C_1 :=$ **KargerStein**(G_t) // inde-
- $C_2 :=$ **KargerStein**(G_t) // pendent // runs
- **return** smaller of C_1, C_2

- Idea: stop when a min-cut is still likely to exist and recurse
- After $t = n - n/\sqrt{2} - 1$ steps we have

$$\Pr[\mathcal{E}_t] = \frac{(\cancel{n-n} + n/\sqrt{2} + 1)(\cancel{n-n} + n/\sqrt{2} + \cancel{1-1})}{n(n-1)} = \frac{n^2/2 + n/\sqrt{2}}{n(n-1)} = \frac{\cancel{n}(n/2 + 1/\sqrt{2})}{\cancel{n}(n-1)} = \frac{1}{2} \cdot \underbrace{\frac{n+\sqrt{2}}{n-1}}_{\geq 1} \geq \frac{1}{2}$$

- Probability that no mistake made after t steps still large

Karger-Stein: Running Time

KargerStein$(G_0 = (V_0, E_0))$

	KargerStein$(G_0 = (V_0, E_0))$
// $O(1)$	**if** $\|V_0\| = 2$ **then** return unique cut
// $O(n)$	**for** $i = 1$ to $t = \|V_0\| - \frac{\|V_0\|}{\sqrt{2}} - 1$ **do**
// $O(1)$	$e :=$ **unif**(E_{i-1})
// $O(n)$	$G_i = G_{i-1}.$**contract**(e)
	$C_1 :=$ **KargerStein**(G_t) // inde-
	$C_2 :=$ **KargerStein**(G_t) // pendent // runs
	return smaller of C_1, C_2

Karger-Stein: Running Time

Karlsruhe Institute of Technology

Recursion

- After $t = n - n/\sqrt{2} - 1$ steps the number of nodes is $n/\sqrt{2} + 1$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}} + 1\right) + O(n^2)$$

```
          KargerStein(G_0 = (V_0, E_0))
// O(1)     if |V_0| = 2 then return unique cut
// O(n)     for i = 1 to t = |V_0| - |V_0|/√2 - 1 do
// O(1)       e := unif(E_{i-1})
// O(n)       G_i = G_{i-1}.contract(e)
            C_1 := KargerStein(G_t)   // inde-
            C_2 := KargerStein(G_t)   // pendent
                                      // runs
            return smaller of C_1, C_2
```

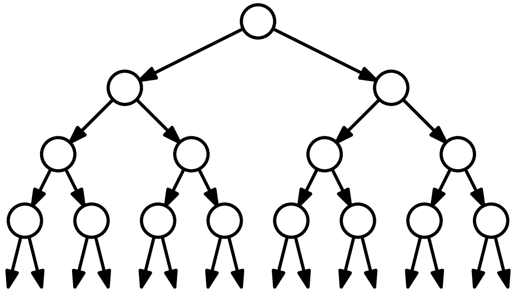
Karger-Stein: Running Time

Recursion

- After $t = n - n/\sqrt{2} - 1$ steps the number of nodes is $n/\sqrt{2} + 1$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}} + 1\right) + O(n^2)$$

Recursion tree



KargerStein$(G_0 = (V_0, E_0))$

- **if** $|V_0| = 2$ **then** return unique cut — // $O(1)$
- **for** $i = 1$ to $t = |V_0| - \frac{|V_0|}{\sqrt{2}} - 1$ **do** — // $O(n)$
  - $e :=$ **unif**(E_{i-1}) — // $O(1)$
  - $G_i = G_{i-1}$.**contract**(e) — // $O(n)$
- $C_1 :=$ **KargerStein**(G_t) — // inde-
- $C_2 :=$ **KargerStein**(G_t) — // pendent // runs
- **return** smaller of C_1, C_2

Karger-Stein: Running Time

Karlsruhe Institute of Technology

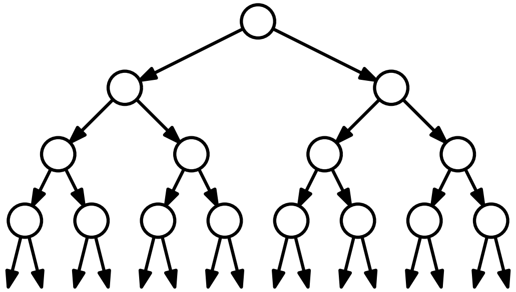
Recursion

- After $t = n - n/\sqrt{2} - 1$ steps the number of nodes is $n/\sqrt{2} + 1$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}} + 1\right) + O(n^2)$$

Recursion tree

- Layers: $\log_{\sqrt{2}}(n)$



KargerStein$(G_0 = (V_0, E_0))$

- // $O(1)$ — **if** $|V_0| = 2$ **then** return unique cut
- // $O(n)$ — **for** $i = 1$ to $t = |V_0| - \frac{|V_0|}{\sqrt{2}} - 1$ **do**
  - // $O(1)$ — $e :=$ **unif**(E_{i-1})
  - // $O(n)$ — $G_i = G_{i-1}$.**contract**(e)
- $C_1 :=$ **KargerStein**(G_t) // inde-
- $C_2 :=$ **KargerStein**(G_t) // pendent // runs
- **return** smaller of C_1, C_2

Karger-Stein: Running Time

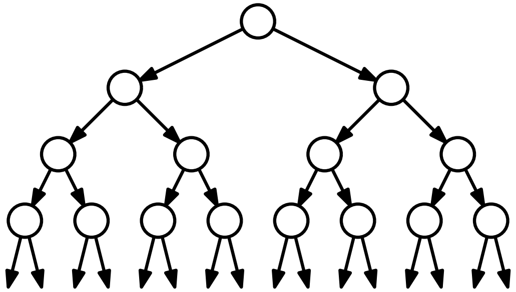
Recursion

- After $t = n - n/\sqrt{2} - 1$ steps the number of nodes is $n/\sqrt{2} + 1$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}} + 1\right) + O(n^2)$$

Recursion tree

- Layers: $\log_{\sqrt{2}}(n)$
- Nodes on layer j: 2^j



KargerStein$(G_0 = (V_0, E_0))$
- $// O(1)$ **if** $|V_0| = 2$ **then** return unique cut
- $// O(n)$ **for** $i = 1$ to $t = |V_0| - \frac{|V_0|}{\sqrt{2}} - 1$ **do**
  - $// O(1)$ $e :=$ **unif**(E_{i-1})
  - $// O(n)$ $G_i = G_{i-1}.$**contract**(e)
- $C_1 :=$ **KargerStein**(G_t) // inde-
- $C_2 :=$ **KargerStein**(G_t) // pendent // runs
- **return** smaller of C_1, C_2

Karger-Stein: Running Time

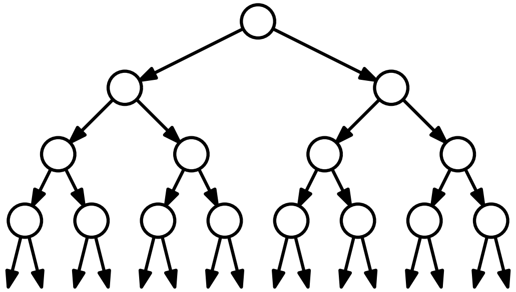
Recursion

- After $t = n - n/\sqrt{2} - 1$ steps the number of nodes is $n/\sqrt{2} + 1$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}} + 1\right) + O(n^2)$$

Recursion tree

- Layers: $\log_{\sqrt{2}}(n)$
- Nodes on layer j: 2^j
- Time on layer j: $O\left(\left(\frac{n}{\sqrt{2}^j}\right)^2\right)$



KargerStein$(G_0 = (V_0, E_0))$
- **if** $|V_0| = 2$ **then** return unique cut // $O(1)$
- **for** $i = 1$ to $t = |V_0| - \frac{|V_0|}{\sqrt{2}} - 1$ **do** // $O(n)$
  - $e :=$ **unif**(E_{i-1}) // $O(1)$
  - $G_i = G_{i-1}$.**contract**(e) // $O(n)$
- $C_1 :=$ **KargerStein**(G_t) // inde-
- $C_2 :=$ **KargerStein**(G_t) // pendent // runs
- **return** smaller of C_1, C_2

Karger-Stein: Running Time

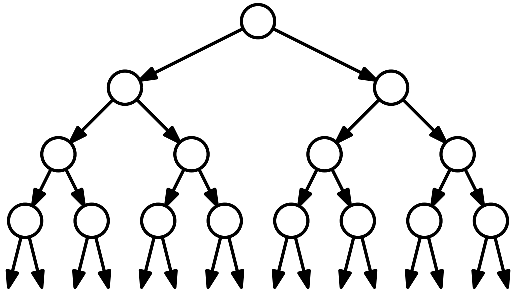
Recursion

- After $t = n - n/\sqrt{2} - 1$ steps the number of nodes is $n/\sqrt{2} + 1$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}} + 1\right) + O(n^2)$$

Recursion tree

- Layers: $\log_{\sqrt{2}}(n)$
- Nodes on layer j: 2^j
- Time on layer j: $O\left(\left(\frac{n}{\sqrt{2}^j}\right)^2\right)$



KargerStein$(G_0 = (V_0, E_0))$

// $O(1)$	**if** $\lvert V_0\rvert = 2$ **then** return unique cut
// $O(n)$	**for** $i = 1$ to $t = \lvert V_0\rvert - \frac{\lvert V_0\rvert}{\sqrt{2}} - 1$ **do**
// $O(1)$	$e :=$ **unif**(E_{i-1})
// $O(n)$	$G_i = G_{i-1}$.**contract**(e)

$C_1 :=$ **KargerStein**(G_t) // inde-
$C_2 :=$ **KargerStein**(G_t) // pendent // runs
return smaller of C_1, C_2

$$T(n) = \sum_{j=1}^{\log_{\sqrt{2}}(n)} 2^j \cdot O\left(\left(\frac{n}{\sqrt{2}^j}\right)^2\right) = O\left(n^2 \cdot \sum_{j=1}^{\log_{\sqrt{2}}(n)} \frac{2^j}{2^j}\right) = O(n^2 \log_{\sqrt{2}}(n)) = O(n^2 \log(n))$$

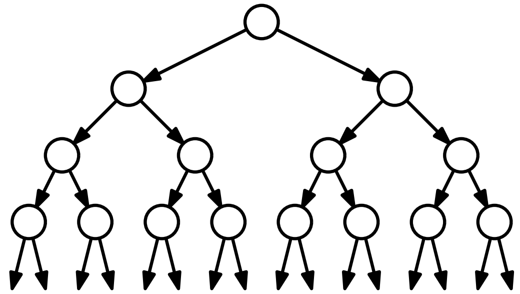
Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

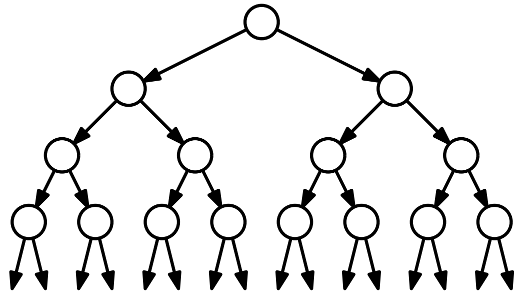


Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

- A node is a successful node if it still contains a min-cut of the original graph

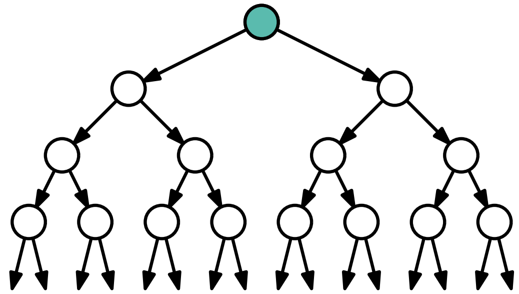


Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

- A node is a successful node if it still contains a min-cut of the original graph

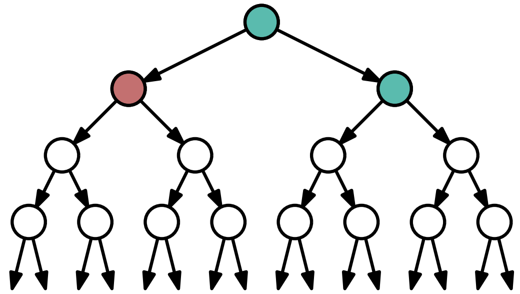


Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

- A node is a successful node if it still contains a min-cut of the original graph

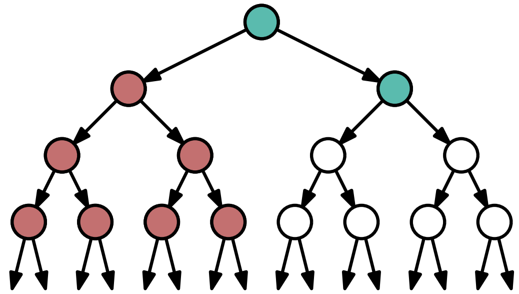


Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

- A node is a successful node if it still contains a min-cut of the original graph

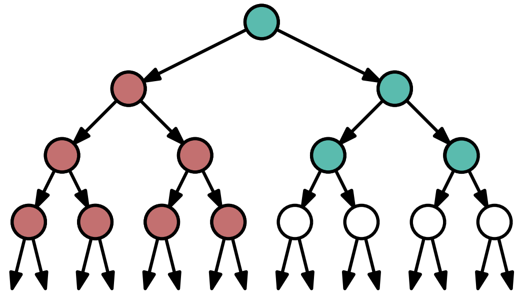


Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

- A node is a successful node if it still contains a min-cut of the original graph

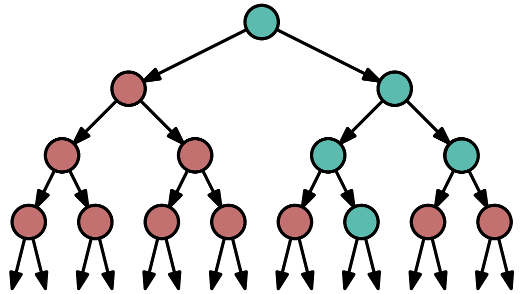


Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

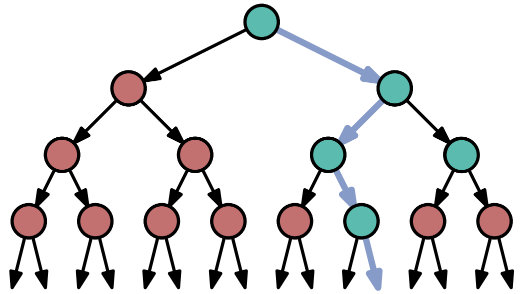
- A node is a successful node if it still contains a min-cut of the original graph



Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

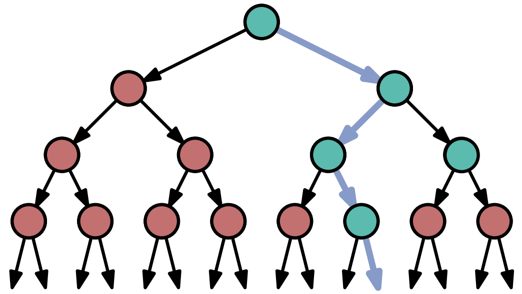


- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

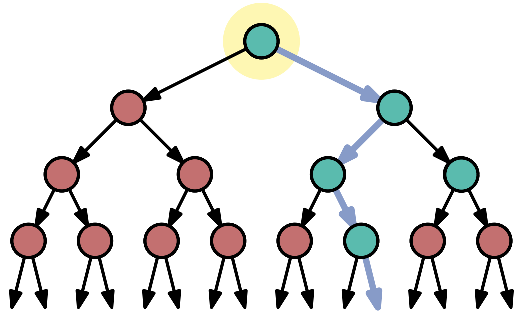


- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

  $\Pr[\mathcal{P}_0] \geq 1/2$

Karger-Stein: Success Probability

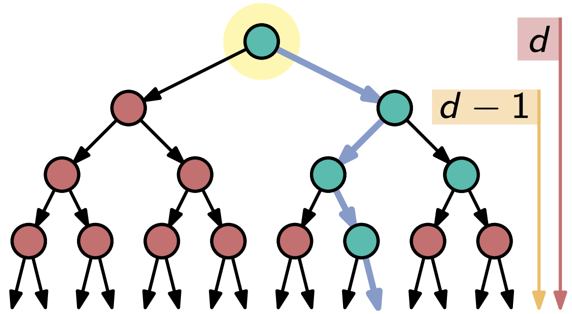
- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree

- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

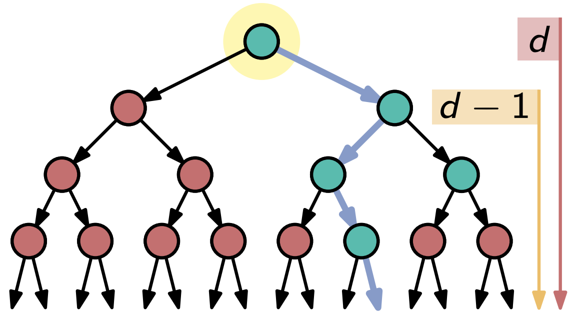
$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right)$



Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

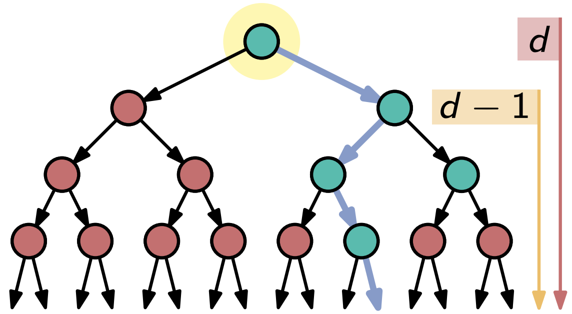
$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - \underbrace{(1 - \Pr[\mathcal{P}_{d-1}])^2}_{\text{no path}}\right)$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

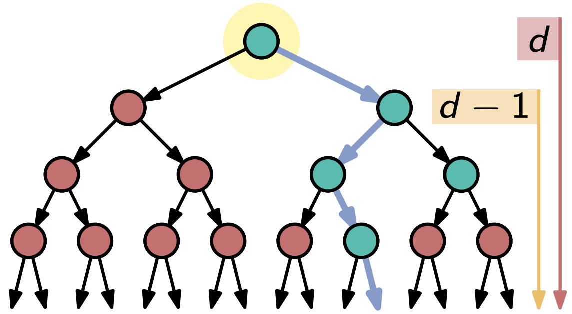
$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \big(1 - \underbrace{\underbrace{(1 - \Pr[\mathcal{P}_{d-1}])}_{\text{no path}}{}^2}_{\text{for left } \textit{and} \text{ right child (independently)}}\big)$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

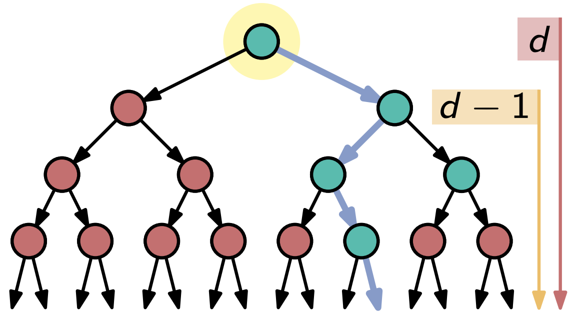
$$\Pr[\mathcal{P}_0] \geq 1/2$$

$$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \underbrace{\left(1 - \underbrace{(1 - \Pr[\mathcal{P}_{d-1}])}_{\text{no path}}\underbrace{{}^2}_{\text{for left } and \text{ right child (independently)}}\right)}_{\text{at least one path among left and right}}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

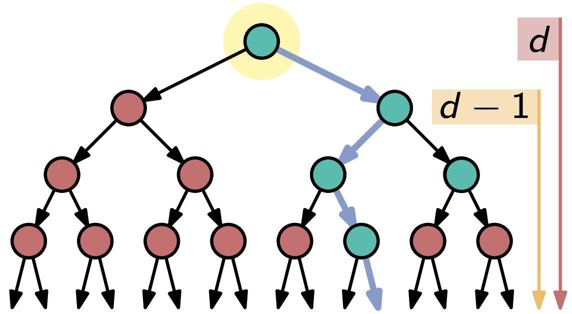
$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right)$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$$\Pr[\mathcal{P}_0] \geq 1/2$$

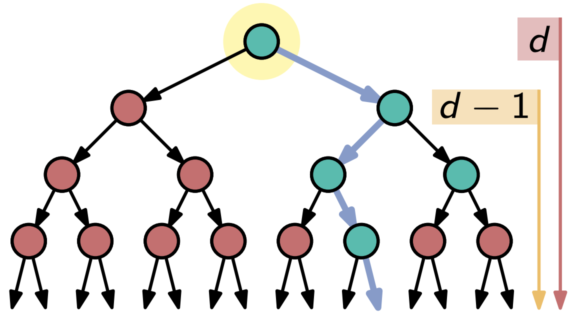
$$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right)$$

$$= \frac{1}{2} \cdot \left(1 - \left(1 - 2\Pr[\mathcal{P}_{d-1}] + \Pr[\mathcal{P}_{d-1}]^2\right)\right)$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

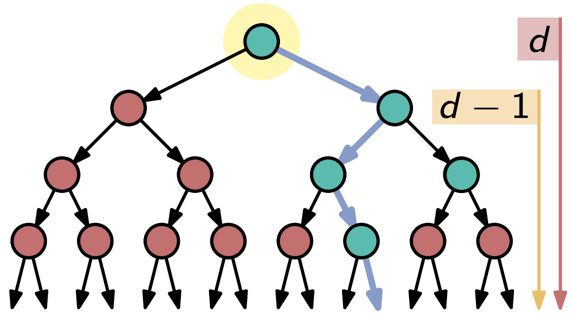
$$\Pr[\mathcal{P}_0] \geq 1/2$$

$$\begin{aligned}\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) &\geq \tfrac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \\ &= \tfrac{1}{2} \cdot \left(\cancel{1} - (\cancel{1} - 2\Pr[\mathcal{P}_{d-1}] + \Pr[\mathcal{P}_{d-1}]^2)\right) \\ &= \tfrac{1}{2} \cdot \left(2\Pr[\mathcal{P}_{d-1}] - \Pr[\mathcal{P}_{d-1}]^2\right)\end{aligned}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

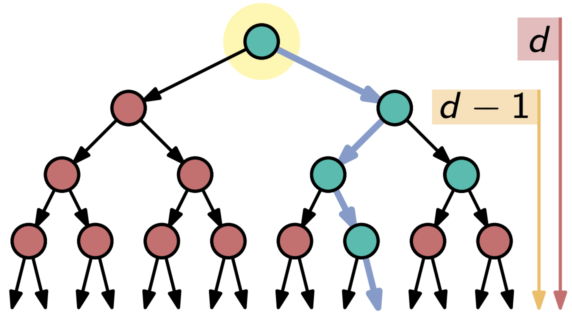
$$\Pr[\mathcal{P}_0] \geq 1/2$$

$$\begin{aligned}
\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) &\geq \tfrac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \\
&= \tfrac{1}{2} \cdot \left(\cancel{1} - (\cancel{1} - 2\Pr[\mathcal{P}_{d-1}] + \Pr[\mathcal{P}_{d-1}]^2)\right) \\
&= \tfrac{1}{2} \cdot \left(2\Pr[\mathcal{P}_{d-1}] - \Pr[\mathcal{P}_{d-1}]^2\right) \\
&= \Pr[\mathcal{P}_{d-1}] - \tfrac{1}{2}\Pr[\mathcal{P}_{d-1}]^2
\end{aligned}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

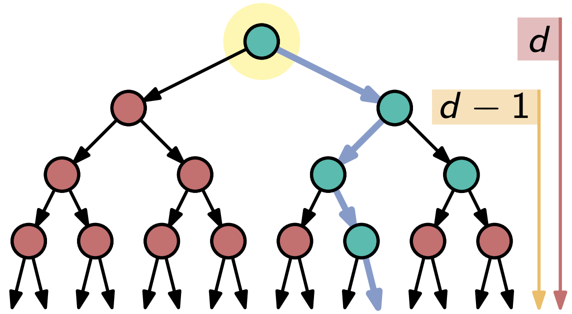
$$\Pr[\mathcal{P}_0] \geq 1/2$$

$$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \tfrac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

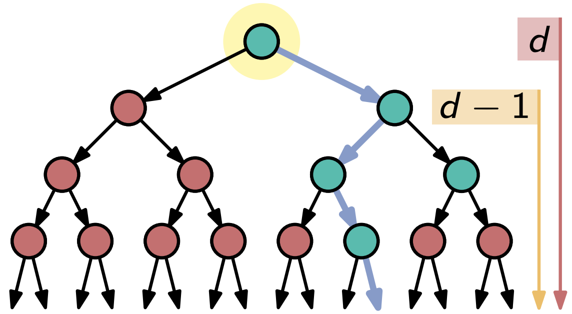
$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

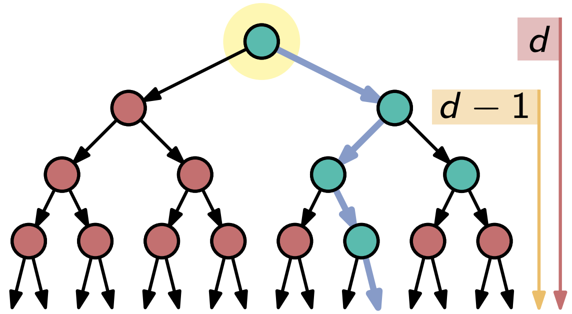
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

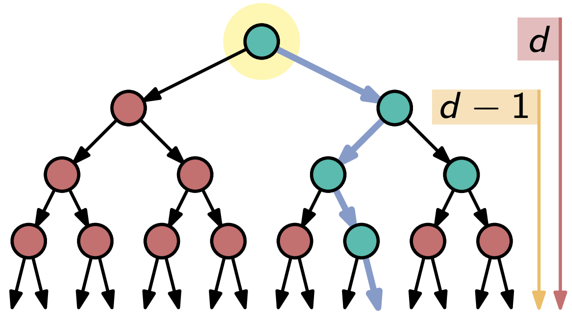
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

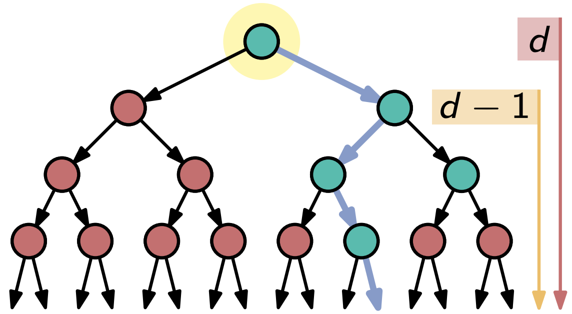
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

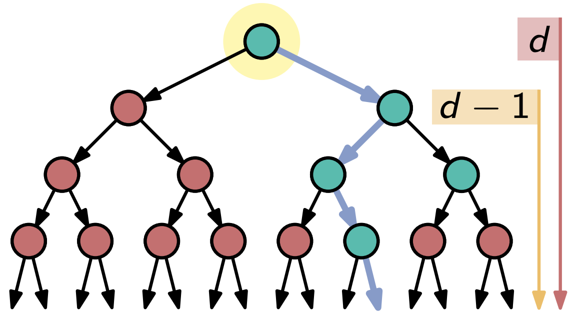
- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

  $\geq \frac{1}{d+1} - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

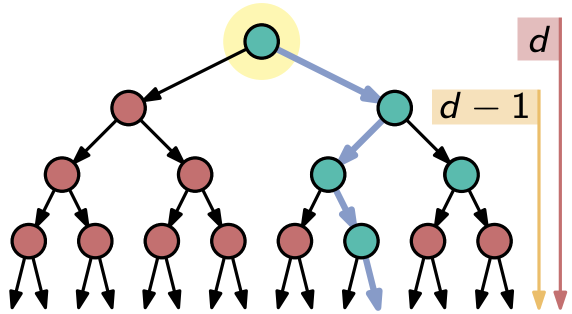
$\geq \frac{1}{d+1} - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

?

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

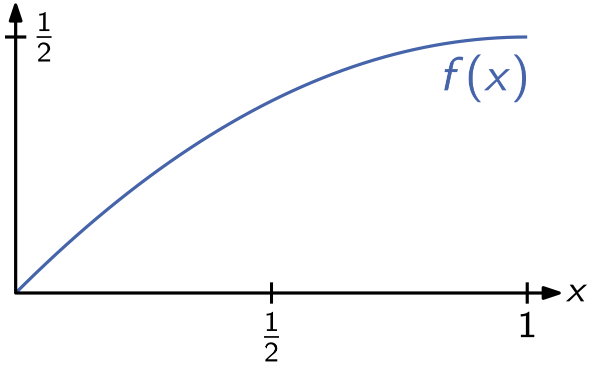
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

$\geq \frac{1}{d+1} \boxed{- \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2}$

?

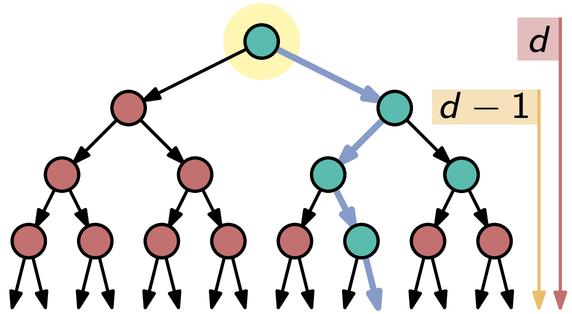
$f(x) = x - \frac{1}{2}x^2$



Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

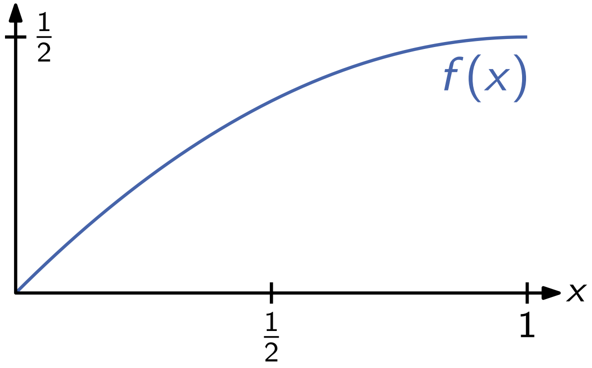
- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

$\geq \frac{1}{d+1} - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

?

$f(x) = x - \frac{1}{2}x^2$

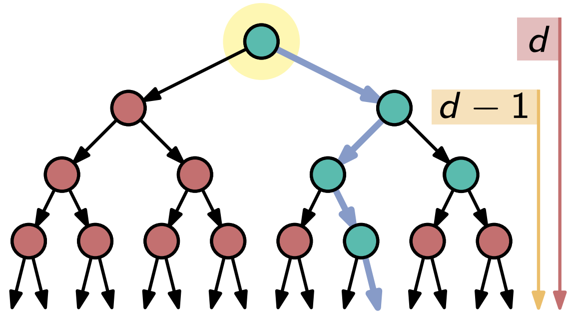
$f(x)\frac{\mathrm{d}}{\mathrm{d}x} = 1 - x$



Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$$\Pr[\mathcal{P}_0] \geq 1/2$$

$$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \tfrac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \tfrac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

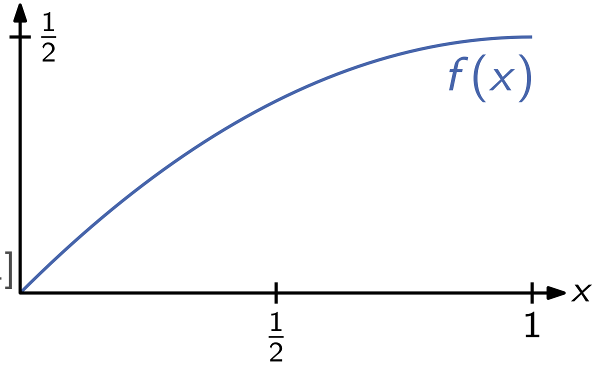
$$\geq \frac{1}{d+1} \boxed{- \tfrac{1}{2}\Pr[\mathcal{P}_{d-1}]^2}$$

?

$$f(x) = x - \tfrac{1}{2}x^2$$

$$f(x)\frac{\mathrm{d}}{\mathrm{d}x} = \underbrace{1 - x}_{\geq 0}$$

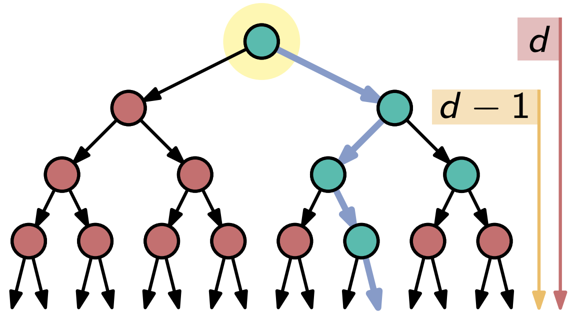
for $x \in [0, 1]$



Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

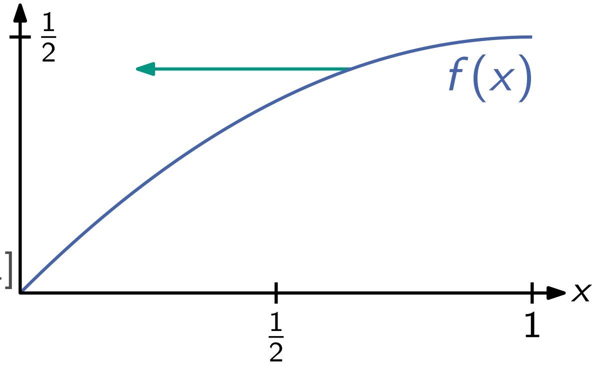
$\geq \frac{1}{d+1} \boxed{- \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2}$

?

$f(x) = x - \frac{1}{2}x^2$

$f(x)\frac{\mathrm{d}}{\mathrm{d}x} = \underbrace{1 - x}_{\geq 0}$

for $x \in [0, 1]$



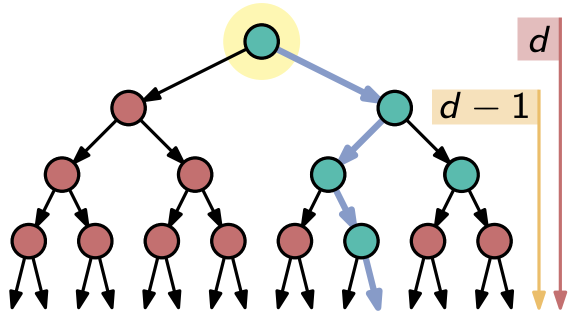
smaller x

Karger-Stein: Success Probability

Karlsruhe Institute of Technology

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$$\Pr[\mathcal{P}_0] \geq 1/2$$

$$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \tfrac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \tfrac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

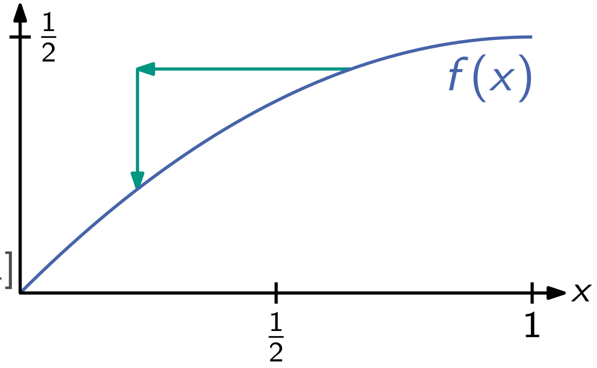
$$\geq \frac{1}{d+1} - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$$

?

$$f(x) = x - \tfrac{1}{2}x^2$$

$$f(x)\frac{\mathrm{d}}{\mathrm{d}x} = \underbrace{1 - x}_{\geq 0}$$

for $x \in [0, 1]$

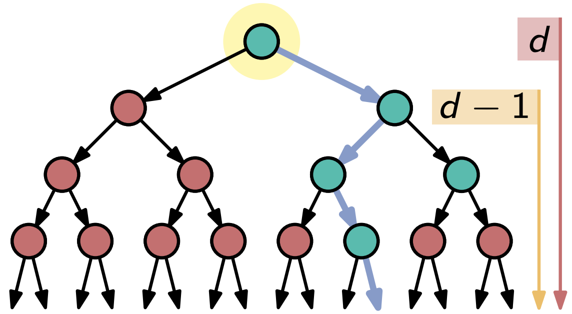


smaller x yields smaller $f(x)$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

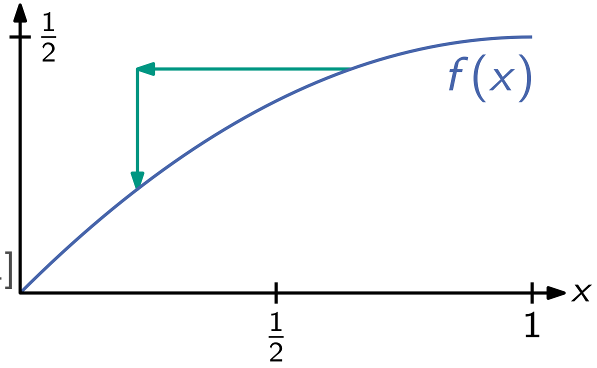
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$
  $\geq \frac{1}{d+1} - \frac{1}{2}\left(\frac{1}{d+1}\right)^2$

$f(x) = x - \frac{1}{2}x^2$

$f(x)\frac{\mathrm{d}}{\mathrm{d}x} = \underbrace{1 - x}_{\geq 0}$

for $x \in [0, 1]$

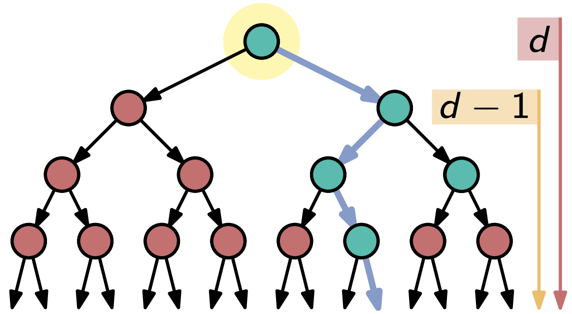


smaller x yields smaller $f(x)$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

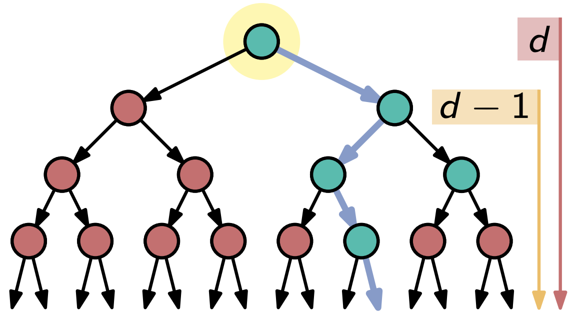
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$
  $\geq \frac{1}{d+1} - \frac{1}{2}\left(\frac{1}{d+1}\right)^2$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

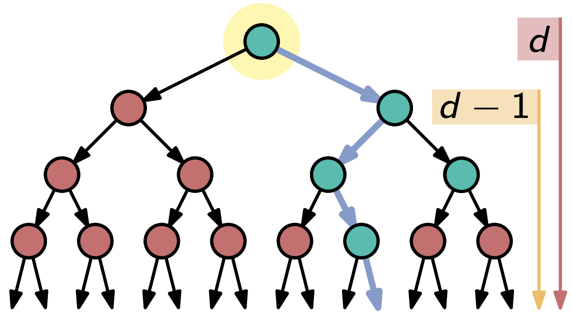
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$
  $\geq \frac{1}{d+1} - \frac{1}{2(d+1)(d+1)}$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

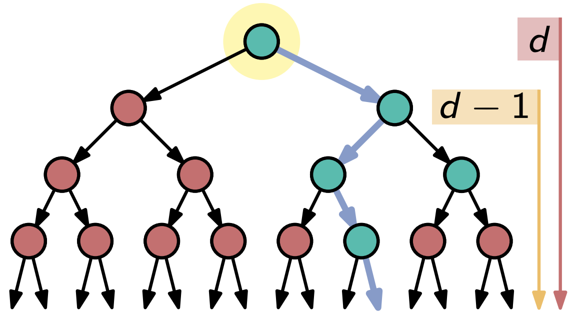
Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$
  $\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

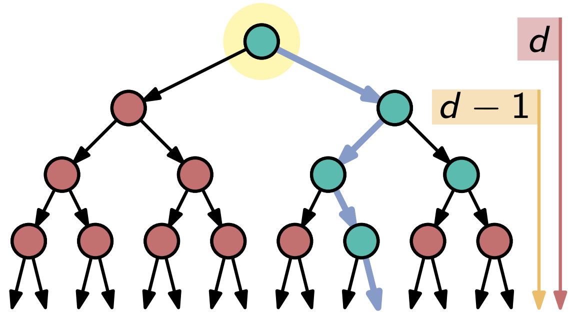
$2d \geq d$ for $d \geq 0$

$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

$2d \geq d$ for $d \geq 0$

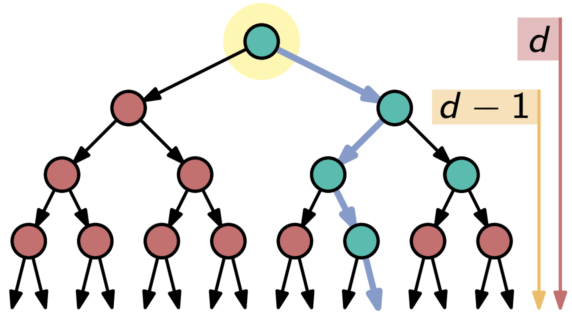
$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

$$= \frac{d+2}{(d+1)(d+2)} - \frac{1}{(d+2)(d+1)}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

$2d \geq d$
for $d \geq 0$

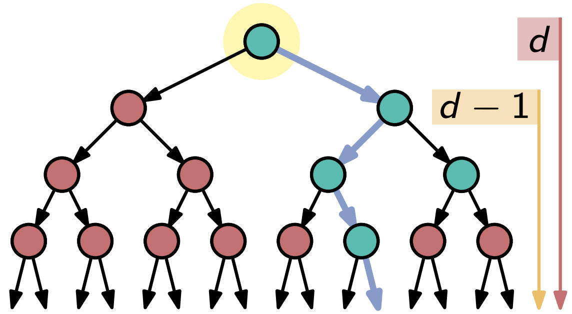
$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

$$= \frac{d+2-1}{(d+1)(d+2)}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

$2d \geq d$ for $d \geq 0$

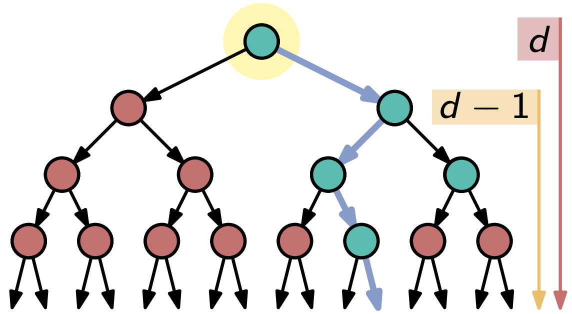
$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

$$= \frac{d+1}{(d+1)(d+2)}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

$2d \geq d$
for $d \geq 0$

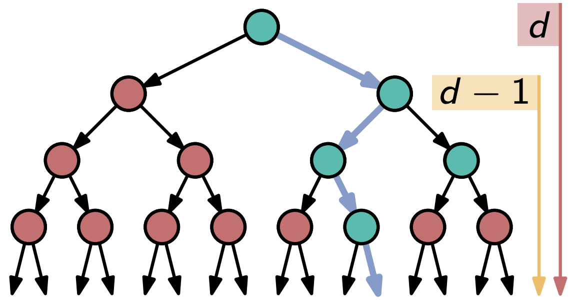
$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

$$= \frac{\cancel{d+1}}{(\cancel{d+1})(d+2)} = \frac{1}{d+2}$$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

$2d \geq d$ for $d \geq 0$

$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

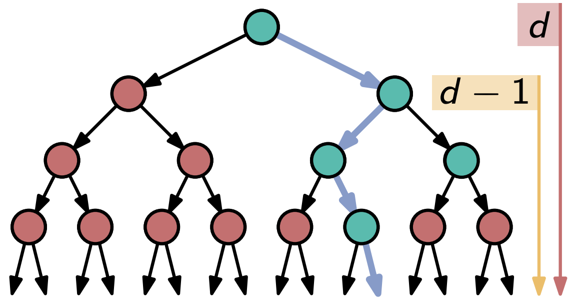
$$= \frac{\cancel{d+1}}{(\cancel{d+1})(d+2)} = \frac{1}{d+2}$$

- $\Pr[\text{“min-cut on layer } d\text{”}] \geq \frac{1}{d+2}$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2}\Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

$2d \geq d$
for $d \geq 0$

$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

$$= \frac{\cancel{d+1}}{(\cancel{d+1})(d+2)} = \frac{1}{d+2}$$

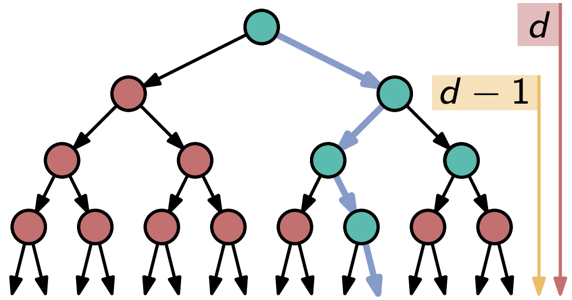
- $\Pr[\text{“min-cut on layer } d\text{”}] \geq \frac{1}{d+2}$
- How many layers in the tree?

$\to \log_{\sqrt{2}}(n)$

Karger-Stein: Success Probability

- After $t = n - n/\sqrt{2} - 1$ steps we have $\Pr[\mathcal{E}_t] \geq 1/2$ (t was chosen to achieve exactly that)

Recursion tree



- A node is a successful node if it still contains a min-cut of the original graph
- A path is a successful path if it contains only successful nodes
- $\mathcal{P}_d$: there exists a successful path of length d starting at the root

$\Pr[\mathcal{P}_0] \geq 1/2$

$\Pr[\mathcal{P}_d] = \Pr[\mathcal{P}_0] \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) \geq \frac{1}{2} \cdot \left(1 - (1 - \Pr[\mathcal{P}_{d-1}])^2\right) = \Pr[\mathcal{P}_{d-1}] - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

Claim $\Pr[\mathcal{P}_d] \geq \frac{1}{d+2}$ (proof via induction)

- Base case $d = 0$: $\Pr[\mathcal{P}_0] \geq 1/2$, Assumption: $\Pr[\mathcal{P}_{d-1}] \geq \frac{1}{d+1}$
- Step: $\Pr[\mathcal{P}_d] \geq \Pr[\mathcal{P}_{d-1}] - \frac{1}{2} \Pr[\mathcal{P}_{d-1}]^2$

$$\geq \frac{1}{d+1} - \frac{1}{(2d+2)(d+1)}$$

$2d \geq d$ for $d \geq 0$

$$\geq \frac{1}{d+1} - \frac{1}{(d+2)(d+1)}$$

$$= \frac{\cancel{d+1}}{(\cancel{d+1})(d+2)} = \frac{1}{d+2}$$

- $\Pr[\text{"min-cut on layer } d\text{"}] \geq \frac{1}{d+2}$
- How many layers in the tree?
  $\to \log_{\sqrt{2}}(n)$
- $\Pr[\text{"min-cut returned"}] \geq \frac{1}{O(\log(n))}$

Karger-Stein Amplified

Theorem: On a graph with n nodes, Karger-Stein runs in $O(n^2 \log(n))$ time and returns a minimum cut with probability at least $1/O(\log(n))$.

Karger-Stein Amplified

Theorem: On a graph with n nodes, Karger-Stein runs in $O(n^2 \log(n))$ time and returns a minimum cut with probability at least $1/O(\log(n))$.

Reminder: Karger $\to 1/O(n^2)$ in $O(n^2)$ time

Karger-Stein Amplified

Theorem: On a graph with n nodes, Karger-Stein runs in $O(n^2 \log(n))$ time and returns a minimum cut with probability at least $1/O(\log(n))$.

Reminder: Karger $\rightarrow 1/O(n^2)$ in $O(n^2)$ time

Amplification

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Karger-Stein Amplified

Theorem: On a graph with n nodes, Karger-Stein runs in $O(n^2 \log(n))$ time and returns a minimum cut with probability at least $1/O(\log(n))$.

Reminder: Karger $\rightarrow 1/O(n^2)$ in $O(n^2)$ time

Amplification

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{t}{O(\log(n))}\right) = 1 - O\left(\frac{1}{n}\right)$$

for $t = \log^2(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Karger-Stein Amplified

Theorem: On a graph with n nodes, Karger-Stein runs in $O(n^2 \log(n))$ time and returns a minimum cut with probability at least $1/O(\log(n))$.

Reminder: Karger $\to 1/O(n^2)$ in $O(n^2)$ time

Amplification

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{t}{O(\log(n))}\right) \underset{\substack{\uparrow \\ \text{for } t = \log^2(n)}}{=} 1 - O\left(\frac{1}{n}\right)$$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

Corollary: On a graph with n nodes, $O(\log^2(n))$ repetitions of Karger-Stein run in $O(n^2 \log^3(n))$ total time and return a minimum cut with high probability.

Karger-Stein Amplified

Theorem: On a graph with n nodes, Karger-Stein runs in $O(n^2 \log(n))$ time and returns a minimum cut with probability at least $1/O(\log(n))$.

Reminder: Karger $\to 1/O(n^2)$ in $O(n^2)$ time

Amplification

$$\Pr[\text{“min-cut found”}] \geq 1 - \exp\left(-\frac{t}{O(\log(n))}\right) = 1 - O\left(\frac{1}{n}\right)$$

for $t = \log^2(n)$

Success probability $\geq p$
Number of repetitions t
Amplified prob. $\geq 1 - e^{-pt}$

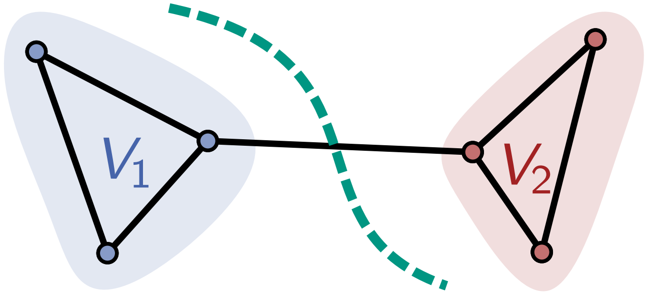
Corollary: On a graph with n nodes, $O(\log^2(n))$ repetitions of Karger-Stein run in $O(n^2 \log^3(n))$ total time and return a minimum cut with high probability.

- Compared to $O(n^4 \log(n))$ for Karger
- Compared to $\Omega(n^3)$ for deterministic approaches

Conclusion

Cuts

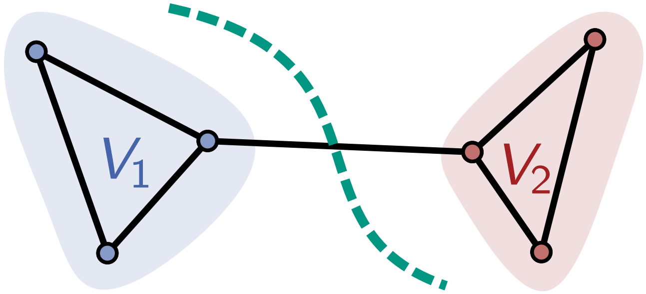
- Fundamental graph problem
- Many deterministic flow-based algorithms
- . . . with worst-case running times in $\Omega(n^3)$



Conclusion

Cuts
- Fundamental graph problem
- Many deterministic flow-based algorithms
- ... with worst-case running times in $\Omega(n^3)$

Randomized Algorithms
- Simple randomized cut via reservoir sampling
- Karger's edge-contraction algorithm



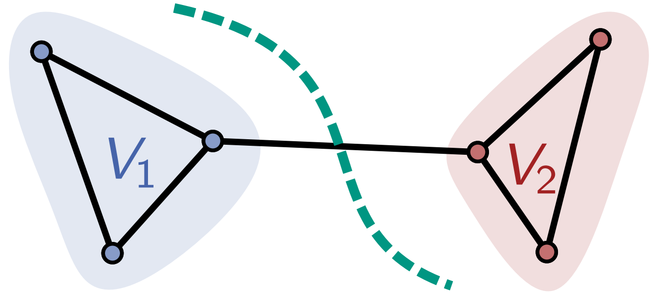
Assumptions: We can sample ...
- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

Conclusion

Cuts

- Fundamental graph problem
- Many deterministic flow-based algorithms
- . . . with worst-case running times in $\Omega(n^3)$



Randomized Algorithms

- Simple randomized cut via reservoir sampling
- Karger's edge-contraction algorithm

Assumptions: We can sample . . .
- uniformly from $\{0, \ldots, O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

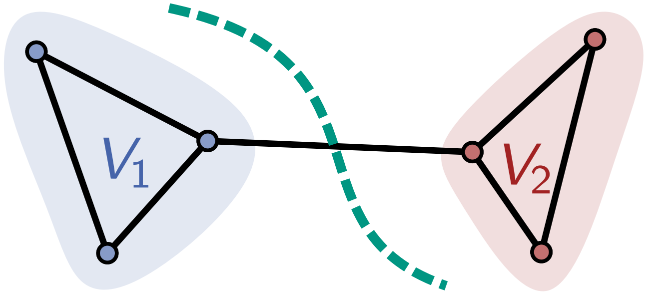
Not possible in theory. Reasonable in practice.

Probability Amplification

- Monte Carlo algorithms with and without biases
- Repetitions amplify success probability
- Karger-Stein: Amplify before failure probability gets large

Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Conclusion



Cuts

- Fundamental graph problem
- Many deterministic flow-based algorithms
- ... with worst-case running times in $\Omega(n^3)$

Randomized Algorithms

- Simple randomized cut via reservoir sampling
- Karger's edge-contraction algorithm

Assumptions: We can sample ...

- uniformly from $\{0, ..., O(n+m)\}$ in $O(1)$ time
- uniformly from $[0, 1]$ in $O(1)$ time

Not possible in theory. Reasonable in practice.

Probability Amplification

- Monte Carlo algorithms with and without biases
- Repetitions amplify success probability
- Karger-Stein: Amplify before failure probability gets large

Algo Output \ Correct Answer	✗	✓
✗	true neg	false neg
✓	false pos	true pos

Outlook

"Minimum cuts in near-linear time", Karger, J.Acm. '00

Success w.h.p. in time $O(m \log^3(n))$

"Faster algorithms for edge connectivity via random 2-out contractions", Ghaffari & Nowicki & Thorup, SODA'20

Success w.h.p. in time $O(m \log(n))$ and $O(m + n \log^3(n))$