Randomized Computation

- Deterministic and (Las Vegas & Monte Carlo) Randomized Algorithms
- Probability Review
- Probabilistic Analysis of deterministic QUICK-SORT Algorithm
- RANDOMIZED-SELECT and RANDOMIZED-QUICK-SORT
- Max-Cut
- Min-Cut
- MAX-3-SAT and Derandomization
- Closest Pair
- Randomized Complexity Classes

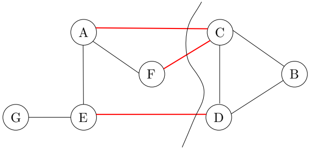
IMDAD ULLAH KHAN

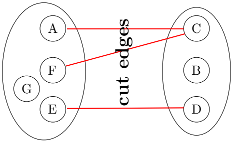
Cuts in Graphs

- Cuts in graphs are useful structures
- Application in network flows, statistical physics, circuit design, complexity and approximation theory

A cut in G is a subset $S \subset V$

- Denoted as $[S, \overline{S}]$
- $S = \emptyset$ and $S = V$ are trivial cuts, we assume that $\emptyset \neq S \neq V$
- A graph on n vertices has 2^n cuts
- An edge (u, v) is crossing the cut $[S, \overline{S}]$, if $u \in S$ and $v \in \overline{S}$



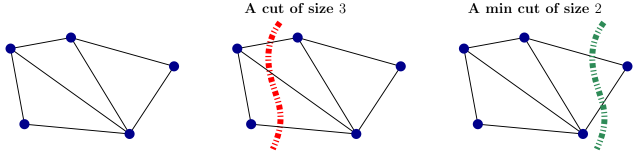


The MIN-CUT problem

A cut in G is a subset $S \subset V$

- Denoted as $[S, \overline{S}]$
- An edge (u, v) is crossing the cut $[S, \overline{S}]$, if $u \in S$ and $v \in \overline{S}$

Size (or cost) of a cut in the number of crossing edges



- In weighted graph size of cut is the sum of weights of crossing edges

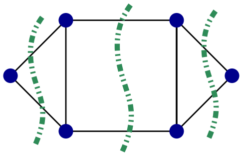
The MIN-CUT(G) problem: Find a cut in G of minimum size?

The MIN-CUT problem

A cut in G is a subset $S \subset V$

- Denoted as $[S, \overline{S}]$
- An edge (u, v) is crossing the cut $[S, \overline{S}]$, if $u \in S$ and $v \in \overline{S}$

Size (or cost) of a cut in the number of crossing edges



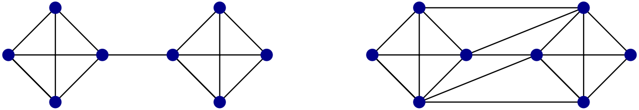
- Min cut does not have to be unique
- size of min-cut is at most the minimum degree of any vertex

The MIN-CUT problem

The MIN-CUT(G) problem: Find a cut in G of minimum size?

Also called Global Min-Cut

Min-cut has applications in network reliability and robustness analysis



The network on the left is easier to disconnect

Normalized min-cut: Spectral clustering applied to image segmentation

Image Segmentation application of MIN-CUT

Separate foreground from background (e.g aircraft/missile from horizon)

https://stackoverflow.com/

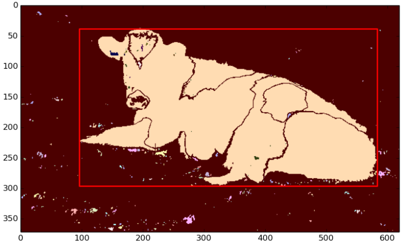


Image Segmentation application of MIN-CUT

Separate foreground from background (e.g aircraft/missile from horizon)

If pixel (x, y) is background/foreground, then so are nearby pixels

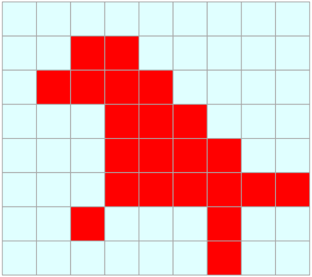


Image Segmentation application of MIN-CUT

Separate foreground from background (e.g aircraft/missile from horizon)

If pixel (x, y) is background/foreground, then so are nearby pixels

Make a graph with nodes for each pixel adjacent to neighboring pixels

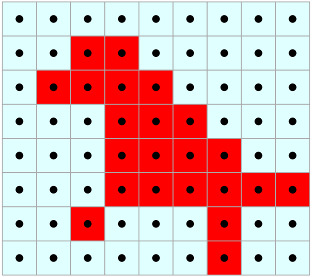


Image Segmentation application of MIN-CUT

Separate foreground from background (e.g aircraft/missile from horizon)

If pixel (x, y) is background/foreground, then so are nearby pixels

Make a graph with nodes for each pixel adjacent to neighboring pixels

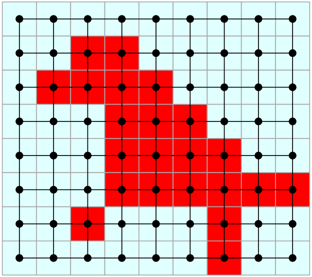


Image Segmentation application of MIN-CUT

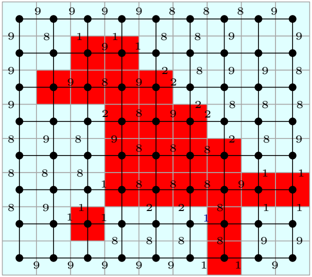
Separate foreground from background (e.g aircraft/missile from horizon)

If pixel (x, y) is background/foreground, then so are nearby pixels

Make a graph with nodes for each pixel adjacent to neighboring pixels

weight of edge (i, j) is p_{ij} a penalty of classifying i and j differently

p_{ij} is a "similarity measure" determined by image processing



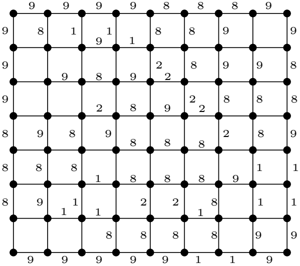
Separate foreground from background (e.g aircraft/missile from horizon)

If pixel (x, y) is background/foreground, then so are nearby pixels

Make a graph with nodes for each pixel adjacent to neighboring pixels

weight of edge (i, j) is p_{ij} a penalty of classifying i and j differently

p_{ij} is a "similarity measure" determined by image processing



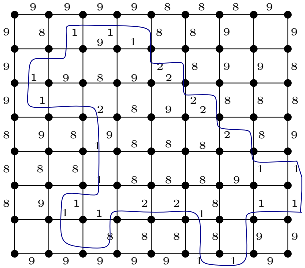
Separate foreground from background (e.g aircraft/missile from horizon)

If pixel (x, y) is background/foreground, then so are nearby pixels

Make a graph with nodes for each pixel adjacent to neighboring pixels

weight of edge (i, j) is p_{ij} a penalty of classifying i and j differently

p_{ij} is a "similarity measure" determined by image processing



Find a min-cut in this weighted graph

Algorithms for Min-Cut

Many deterministic algorithms have been proposed

- Stoer-Wagner $O(nm + n^2 \log m)$ time algorithm
- We study a simple randomized algorithm by Karger
- And an elegant extension of it due to Karger and Stein

These algorithms are based on the Edge Contraction Operation

Types of Graphs: PseudoGraphs and Multigraphs

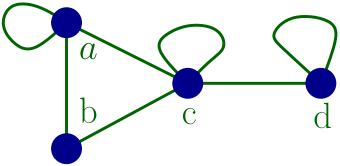
- PseudoGraphs

$G = (V, E)$

V is set of vertices

E is set of edges

(self loops allowed)



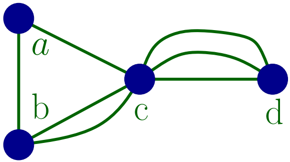
- Multigraphs

$G = (V, E)$

V is set of vertices

E is multi-set of edges

may have self loops too

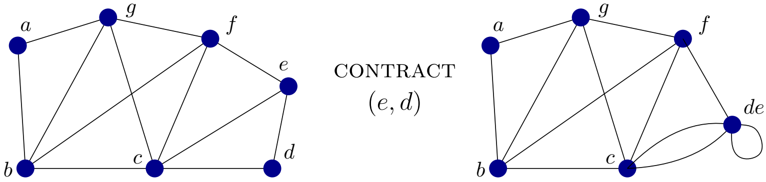


Edge Contraction

Contraction of an edge (u, v) in G constructs a graph $G \setminus uv$

- u and v become one vertex uv
- edge (u, v) becomes a self-loop (we remove it)
- All edges incident on u or v become incident on uv

The resulting graph may become a multigraph (we keep all edges)

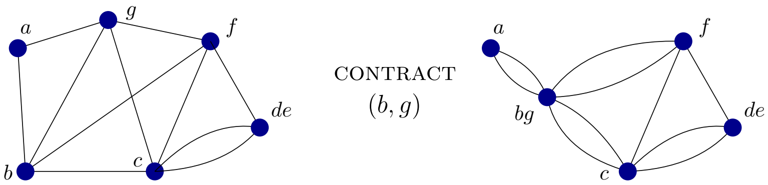


Edge Contraction

Contraction of an edge (u, v) in G constructs a graph $G \setminus uv$

- u and v become one vertex uv
- edge (u, v) becomes a self-loop (we remove it)
- All edges incident on u or v become incident on uv

The resulting graph may become a multigraph (we keep all edges)

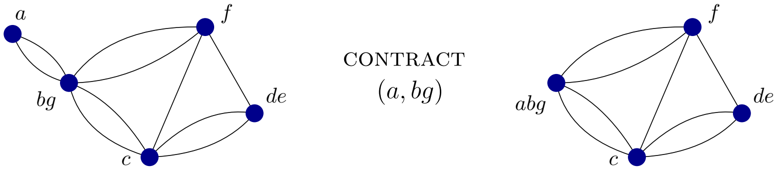


Edge Contraction

Contraction of an edge (u, v) in G constructs a graph $G \setminus uv$

- u and v become one vertex uv
- edge (u, v) becomes a self-loop (we remove it)
- All edges incident on u or v become incident on uv

The resulting graph may become a multigraph (we keep all edges)

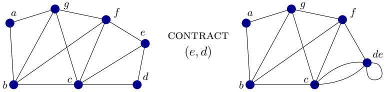


Edge Contraction

Contraction of an edge (u, v) in G constructs a graph $G \setminus uv$

- u and v become one vertex uv
- edge (u, v) becomes a self-loop (we remove it)
- All edges incident on u or v become incident on uv

The resulting graph may become a multigraph (we keep all edges)

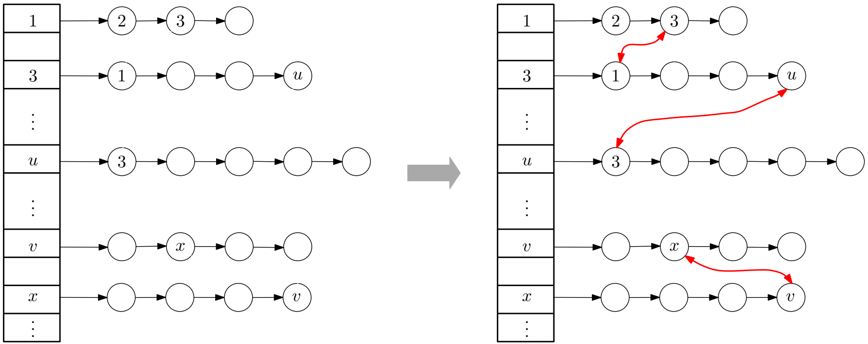


▷ Multigraphs can be saved with multiplicity as edge weight

Edge Contraction: Runtime

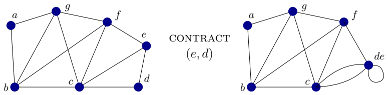
Edge contraction can be performed in $O(n)$ time

- Merge adjacency lists of u and v
- Adjacency lists of other vertices can be updated in $O(n)$ time (if we keep corresponding pointers at entries of adjacency lists)



Edge Contraction

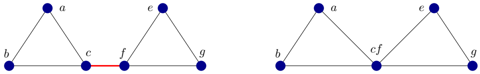
- Contraction of an edge (u, v) in G makes multigraph $G \setminus uv$
- u, v merged into uv, edges incident on u or v become incident on uv



What happens to min cut after contraction?

▷ If the min-cut in G is of size 10, can $G \setminus uv$ have min cut of size 9?

- The min cut in $G \setminus uv$ is at least as large as min cut in G
  - Because any cut in $G \setminus uv$ is "actually" a cut in G too
- The converse is not necessarily true



Edge contraction increases min cut if the edge is in all possible min cuts

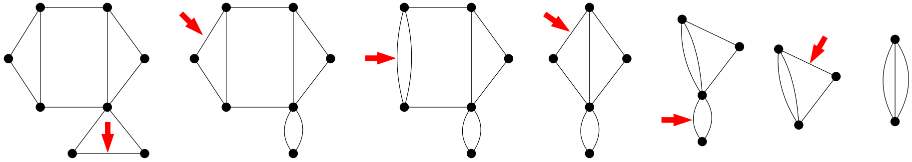
Algorithm : Karger's algorithm for mincut (G)

while there are more than two vertices left in G **do**

  Pick a random edge $e = (u, v)$

  $G \leftarrow G \setminus uv$

return G ▷ the cut induced by the remaining two (super)nodes



A run of Karger algorithm that produces a sub-optimal cut (with 3 edges)

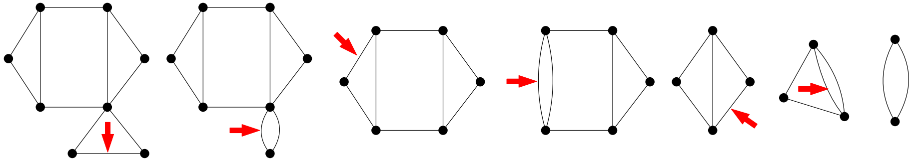
Algorithm : Karger's algorithm for mincut (G)

while there are more than two vertices left in G **do**

  Pick a random edge $e = (u, v)$

  $G \leftarrow G \setminus uv$

return G $\triangleright$ the cut induced by the remaining two (super)nodes



A run of Karger algorithm that produces an optimal cut (with 2 edges)

Karger's Algorithm: Runtime

Algorithm : Karger's algorithm for mincut (G)

while there are more than two vertices left in G **do**

  Pick a random edge $e = (u, v)$

  $G \leftarrow G \setminus uv$

return G ▷ the cut induced by the remaining two (super)nodes

- With the right data structure a contraction can be done in $O(n)$
- Each contraction reduces the number of vertices by 1
- Number of contraction is $n - 2$
- Total runtime is $O(n^2)$

The intuition:

- Let $C = \left[S, \overline{S}\,\right]$ be a specific cut
- If during the execution some edge in C is contracted, the algorithm will not output the cut C
  - If $(u, v) \in C \;\leftrightarrow\; u \in S \wedge v \in \overline{S}$ is contracted, then u and v will belong to the same supernode and (u, v) cannot be a crossing edge
- The algorithm will output C if it never contracts any edge in C

Among all cuts, min-cuts have the least probability of having an edge contracted

Karger's Algorithm: Analysis

Let $G_0 = (V_0, E_0) = G = (V, E)$ $\quad\triangleright\ |V_i| = n_i,\ |E_i| = m_i$

For $0 \leq i \leq n-2$, $G_i = (V_i, E_i)$: graph after ith contraction $\quad\triangleright\ n_i = n - i$

Let $C = \left[S, \overline{S}\right]$ be a (specific) min-cut of size k

Every vertex has degree $\geq k \implies m_0 \geq {}^{kn_0}\!/_2$ $\quad\triangleright \because C$ is a min-cut of size k

C has survived up to $G_i \implies m_i \geq {}^{kn_i}\!/_2 = {}^{k(n-i)}\!/_2$

$Pr[C$ is "killed" in 1st round$]$ $=$ $Pr[$an edge in C is contracted$]$ $= {}^{k}\!/_{m_0} \leq {}^{2}\!/_{n_0}$

$Pr[C$ survives in 1st round$]$ $=$ $Pr[$no edge in C is contracted$]$ $\geq 1 - {}^{2}\!/_{n_0}$

$Pr[C$ survives in $(i+1)$th round $\mid C$ survived so far$]$ $= 1 - {}^{k}\!/_{m_i} \geq 1 - {}^{2}\!/_{n-i}$

$Pr[C$ survives all rounds$]$ $= \prod_{i=0}^{n-3} Pr[C$ survives round $i+1 \mid C$ survived so far$]$

$Pr[C$ survives all rounds$]$ $=$ $Pr[C$ is the output$]$ $= \prod_{i=0}^{n-3} \frac{n-i-2}{n-i}$

$Pr[C$ is the output$]$ $\geq \frac{n-2}{n} \times \frac{n-3}{n-1} \times \frac{n-4}{n-2} \times \ldots \times \frac{2}{4} \times \frac{1}{3} = \frac{2}{n(n-1)} = {}^{1}\!/\binom{n}{2}$

Karger's Algorithm: Analysis

Let $G_0 = (V_0, E_0) = G = (V, E)$ $\qquad \triangleright \; |V_0| = n, \; |E_0| = m$

Let $C = [S, \overline{S}]$ be a (specific) min-cut of size k

$$Pr[C \text{ is the output}] \simeq 1/n^2$$

This probability is very small   is it?

- There are 2^m cuts, many of them min-cuts, we find one of the min-cuts with probability $1/n^2$
- With repeated trials, we amplify the probability to any desired value

Karger's Algorithm: Analysis

Let $G_0 = (V_0, E_0) = G = (V, E)$ $\triangleright$ $|V_0| = n$, $|E_0| = m$

Let $C = [S, \overline{S}]$ be a (specific) min-cut of size k

$$Pr[C \text{ is the output}] \simeq {}^1/_{n^2}$$

- With repeated trials, we amplify the probability to any desired value

Algorithm Good-Min-Cut(G, M)

Run MIN-CUT(G) M times

Return smallest of these M cuts

Algorithm Min-Cut (G)

while more than two vertices left in G **do**

Pick a random edge $e = (u, v)$

$G \leftarrow G \setminus uv$

return G

Karger's Algorithm: Analysis

Let $G_0 = (V_0, E_0) = G = (V, E)$ $\triangleright$ $|V_0| = n$, $|E_0| = m$

$C = [S, \overline{S}]$: a (specific) min-cut of size k $\triangleright$ $Pr[C \text{ is the output}] \simeq 1/n^2$

Algorithm Good-Min-Cut(G, M)

Run $\textsc{Min-Cut}(G)$ M times

Return smallest of these M cuts

Algorithm Min-Cut (G)

while more than two vertices left in G **do**

Pick a random edge $e = (u, v)$

$G \leftarrow G \setminus uv$

return G

$Pr[\text{all } M \text{ runs fail to output } C] = \prod_{i=1}^{n} Pr[\text{Run } i \text{ fails}] \leq \left(1 - 1/n^2\right)^M$

$\forall\, x \in \mathbb{R}\ \ (1 + x) < e^x$ $\triangleright$ A very useful inequality

$Pr[\textsc{good-min-cut}(G, M) \text{ fails to output } C] \leq e^{-M/n^2}$

$M = cn^2 \log n \implies Pr[\textsc{good-min-cut}(G, M) \text{ outputs } C] \geq 1 - 1/n^c$

Runtime is $O(n^4 \log n)$

Karger-Stein Algorithm

Algorithm Good-Min-Cut(G, M)

Run $\textsc{Min-Cut}(G)$ M times

Return smallest of these M cuts

Algorithm : Min-Cut (G)

while more than two vertices left in G **do**

Pick a random edge $e = (u, v)$

$G \leftarrow G \setminus uv$

return G

$Pr[C$ is "killed" in round $1] = Pr[$an edge in C is contracted$] \; = \; {}^{k}/_{m_0} \; \leq \; {}^{2}/_{n}$

$Pr[C$ is "killed" in round $2 \mid C$ survived round $1] \; = \; {}^{k}/_{m_1} \; \leq \; {}^{2}/_{n-1}$

$Pr[C$ is "killed" in rond $(i+1) | \; C$ survived so far$] \; = \; {}^{k}/_{m_i} \leq {}^{2}/_{n-i}$

$Pr[C$ is "killed" in rond $(n-3) | \; C$ survived so far$] \; \leq \; {}^{2}/_{4}$

$Pr[C$ is "killed" in rond $(n-2) | \; C$ survived so far$] \; \leq \; {}^{2}/_{3}$

Bound on probability of wrong contraction increases in each round

As G gets smaller, repeat increasingly many times to reduce the error probability

$\triangleright$ do not waste time repeating the first "few" iterations

Karger-Stein Algorithm

Algorithm Fast-Cut(G)

if $n \leq 6$ **then**
  return Min-cut (via brute force)
$t \leftarrow \lceil 1 + n/\sqrt{2} \rceil$
$H_1 \leftarrow \text{CONTRACT}(G, t)$
$H_2 \leftarrow \text{CONTRACT}(G, t)$
$C_1 \leftarrow \text{FAST-CUT}(H_1)$
$C_2 \leftarrow \text{FAST-CUT}(H_2)$
return smaller of C_1 and C_2

Algorithm Contract (G, t)

function CONTRACT(G, t)
  while more than t vertices left in G **do**
    Pick a random edge $e = (u, v)$
    $G \leftarrow G \setminus uv$
  return G

- Two independent randomly contracted graphs H_1 and H_2 from G
- When H_1 and H_2 are small, make 4 random contractions
- and so on
- When the graph has less than 6 vertices, return min among all $\sim 2^5$ cuts
- Now we cannot chase a fixed minimum cut C, as both X_1 and X_2 could be min cuts (if successful) and we may choose either

Karger-Stein Algorithm

Algorithm Fast-Cut(G)

> **if** $n \leq 6$ **then**
>   **return** Min-cut (via brute force)
> $t \leftarrow \lceil 1 + n/\sqrt{2} \rceil$
> $H_1 \leftarrow \textsc{contract}(G, t)$
> $H_2 \leftarrow \textsc{contract}(G, t)$
> $C_1 \leftarrow \textsc{fast-cut}(H_1)$
> $C_2 \leftarrow \textsc{fast-cut}(H_2)$
> **return** smaller of C_1 and C_2

Algorithm Contract (G, t)

> **function** $\textsc{contract}(G, t)$
>   **while** more than t vertices left in G **do**
>   Pick a random edge $e = (u, v)$
> $G \leftarrow G \setminus uv$
>   **return** G

Let $T(n)$ be runtime of $\textsc{fast-cut}(G)$ with $|V(G)| = n$

$$T(n) \;=\; \begin{cases} 2T\left(n/\sqrt{2}\right) + O(n^2) & \text{if } n > 6 \\ O(1) & \text{else} \end{cases}$$

$\mathbf{T(n) = O(n^2 \log n)}$ $\quad\triangleright$ master theorem

Karger-Stein Algorithm: Quality

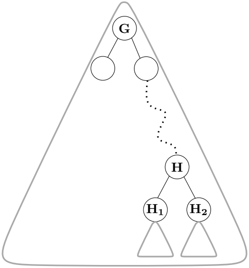
```
function FAST-CUT(G)
    if n ≤ 6 then
        return Min-cut (brute force)
    t ← ⌈1 + n/√2⌉
    H1 ← CONTRACT(G, t)
    H2 ← CONTRACT(G, t)
    C1 ← FAST-CUT(H1)
    C2 ← FAST-CUT(H2)
    return smaller of C1 and C2
```

Algorithm Contract (G, t)

```
function CONTRACT(G, t)
    while more than t vertices left in G do
        Pick a random edge e = (u, v)
        G ← G \ uv
    return G
```

FAST-CUT(G) succeeds iff

1. A min-cut survives the CONTRACT(G,t) step
2. At least one of the FAST-CUT(H_1) and FAST-CUT(H_2) finds a min-cut



Karger-Stein Algorithm: Quality

FAST-CUT(G) succeeds iff

1. A min-cut survives the CONTRACT(G, t) step
2. At least one of the FAST-CUT(H_1) and FAST-CUT(H_2) finds a min-cut

```
1: function FAST-CUT(G)
2:     if n ≤ 6 then
3:         return Min-cut
4:     t ← ⌈1 + n/√2⌉
5:     H1 ← CONTRACT(G, t)
6:     H2 ← CONTRACT(G, t)
7:     C1 ← FAST-CUT(H1)
8:     C2 ← FAST-CUT(H2)
9:     return MIN of C1 and C2
```

Probability a min cut survive CONTRACT(G, t) step ▷ line 5&6

$$Pr[\text{a cut survives } n-t \text{ contractions}] = \prod_{i=0}^{n-t-1} \frac{n-i-2}{n-i}$$

$$= \frac{n-2}{n} \times \ldots \times \frac{t}{t+2} \times \frac{t-1}{t+1} = \frac{t(t-1)}{n(n-1)}$$

$$= {}^{t(t-1)}/_{n(n-1)} \simeq {}^{1}/_{2} \qquad \triangleright\ t = {}^{n}/\sqrt{2}$$

Karger-Stein Algorithm: Quality

FAST-CUT(G) succeeds iff

- A min-cut survives the CONTRACT(G, t) step
- At least one of the FAST-CUT(H_1) and FAST-CUT(H_2) finds a min-cut

```
1: function FAST-CUT(G)
2:    if n ≤ 6 then
3:       return Min-cut
4:    t ← ⌈1 + n/√2⌉
5:    H1 ← CONTRACT(G, t)
6:    H2 ← CONTRACT(G, t)
7:    C1 ← FAST-CUT(H1)
8:    C2 ← FAST-CUT(H2)
9:    return MIN of C1 and C2
```

$P(j)$: prob that FAST-CUT(H) finds min-cut if $|V(H)| = j$

Probability that FAST-CUT(G) succeeds:
$\mathbf{P(n)}$

- A min-cut survives in H_1 (line 5) ▷ Prob: $1/2$
- **AND** C_1 is a min-cut in H_1 (line 7) ▷ Prob: $P(t)$

OR

- A min-cut survives in H_2 (line 6) ▷ Prob: $1/2$
- **AND** C_2 is a min-cut in H_2 (line 8) ▷ Prob: $P(t)$

Karger-Stein Algorithm: Quality

$P(j)$: prob that $\textsc{fast-cut}(H)$ finds min-cut if $|V(H)| = j$

Probability that $\textsc{fast-cut}(G)$ succeeds

$\mathbf{P(n)}$

- A min-cut survives in H_1 (line 5) $\triangleright$ Prob: $1/2$
- **AND** C_1 is a min-cut in H_1 (line 7) $\triangleright$ Prob: $P(t)$

OR

- A min-cut survives in H_2 (line 6) $\triangleright$ Prob: $1/2$
- **AND** C_2 is a min-cut in H_2 (line 8) $\triangleright$ Prob: $P(t)$

$$Pr[\text{Branch-i succeeds}] = Pr\left[\begin{array}{l}\text{A min-cut survives in } H_i \text{ (line 5/6)}\\ \textbf{AND } C_i \text{ is min-cut in } H_i \text{ (line 7/8)}\end{array}\right] = \frac{1}{2}\cdot P(t)$$

$$Pr[\text{Branch-i fails}] = 1 - 1/2P(t) \qquad Pr[\text{Both Branches fail}] = (1 - 1/2P(t))^2$$

$$Pr[\text{Algo succeeds}] = Pr[\textbf{NOT} \text{ Both Branches fail}] \geq 1 - (1 - 1/2P(t))^2$$

Karger-Stein Algorithm: Quality

$P(j)$: prob that $\textsc{fast-cut}(H)$ finds min-cut if $|V(H)| = j$

$$Pr[\text{Branch-i succeeds}] \; = \; Pr\left[\begin{array}{l}\text{A min-cut survives in } H_i \text{ (line 5/6)}\\ \textbf{AND } C_i \text{ is min-cut in } H_i \text{ (line 7/8)}\end{array}\right] \; = \; \frac{1}{2}\cdot P(t)$$

$Pr[\text{Branch-i fails}] \; = \; 1 - {}^1\!/\!{}_2 P(t)$ $\qquad$ $Pr[\text{Both Branches fail}] \; = \; (1 - {}^1\!/\!{}_2 P(t))^2$

$Pr[\text{Algo succeeds}] \; = \; Pr[\textbf{NOT} \text{ Both Branches fail}] \; \geq \; 1 - (1 - {}^1\!/\!{}_2 P(t))^2$

$$P(n) \; \geq \; 1 - (1 - {}^1\!/\!{}_2 P(t))^2 \; = \; 1 - (1 - {}^1\!/\!{}_2 P({}^n\!/\!{}_{\sqrt{2}}))^2 \; = \; \Omega({}^1\!/\!{}_{\log n})$$

Easily proved via induction

Karger-Stein Algorithm: Quality

- FAST-CUT(G) takes $O(n^2 \log n)$ times not much worse than $O(n^2)$ initial version

- Has a success probability $\Omega(1/\log n)$ much better than $\Omega(1/n^2)$ of initial version

- The initial version amplified by $n^2 \log n$ independent trial had runtime $O(n^4 \log n)$ and success probability $\Omega(1 - 1/n^c)$

- FAST-CUT(G) amplified by $c \log^2 n$ independent trial has runtime $O(n^2 \log^3 n)$ and success probability $\Omega(1 - 1/n^c)$