Randomized Algorithms

Minimum cuts by random contraction

Alex Gittens
Rensselaer Polytechnic Institute

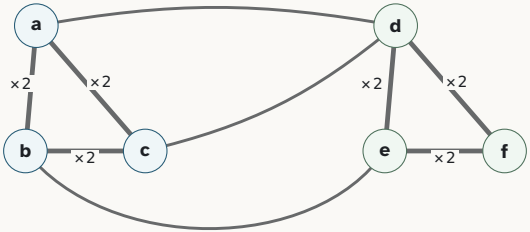
Fall 2026

PART 1

Minimum cuts and random contraction

1

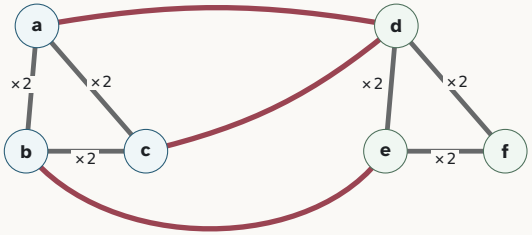
Find the weakest connection



What is the smallest number of edges we can remove to disconnect the graph?

Each ×2 label denotes two parallel edges; every unlabeled segment is one edge.

A minimum cut of size three



Removing the three red edges separates $\{a, b, c\}$ from $\{d, e, f\}$.

To show it is minimum, consider *any* cut. If it separates vertices within either triangle, it crosses at least four internal edge instances (each triangle side has multiplicity two). Otherwise each triangle stays together, and the only nontrivial cut is the displayed three-edge cut.

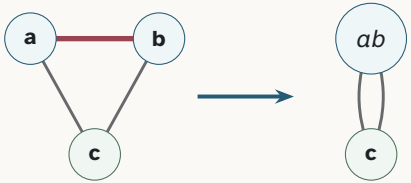
What exactly is a minimum cut?

A **cut** separates the vertices of a connected undirected multigraph $G = (V, E)$ into nonempty sets S and $V \setminus S$.

$$\delta(S) = \{uv \in E : u \in S,\ v \notin S\}, \qquad \lambda = \min_{\emptyset \neq S V} |\delta(S)|.$$

We count parallel edges separately. We want to output a set S attaining λ (or its crossing edges). There are $2^{n-1} - 1$ distinct cuts: put one fixed vertex on the first side, and place each remaining vertex on either side, excluding the trivial choice.

Contracting an edge



Contracting ab replaces a, b by one **supervertex** ab.

Delete self-loops; **keep** the two edges to c.

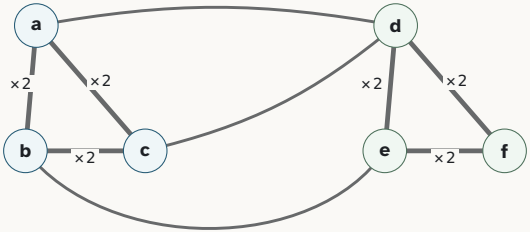
Contraction: the operation

A current vertex (a *supervertex*) represents a set of original vertices. Contracting an edge merges two supervertices.

Input: Current multigraph H and an edge uv
Output: Contracted graph H'
Create supervertex z representing the union of the original vertices in u, v
foreach *other supervertex* x **do**
    replace all ux and vx edges by edges zx
Discard every edge joining u to v (now a self-loop)
return H'

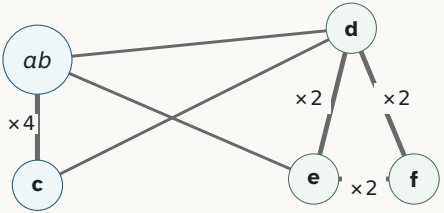
No original edge is discarded except an edge that becomes internal to a supervertex.

A complete contraction run: first move



Suppose we select an edge between a and b. We merge them into ab; the other vertices are unchanged.

A complete contraction run: after the first move

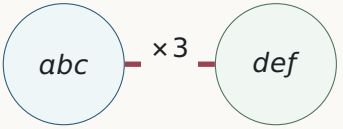


There are five supervertices. The two a–b edge instances become self-loops and disappear. The four ab–c instances remain.

A complete contraction run: finishing

One possible sequence

1. Merge a with b.
2. Merge d with e.
3. Merge ab with c.
4. Merge de with f.



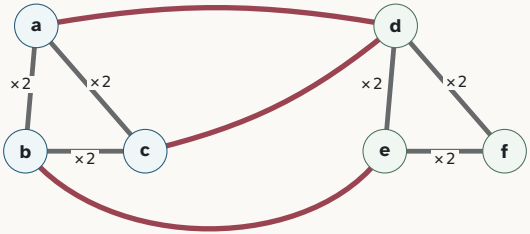
Three edges connect the final two supervertices.

Two supervertices remain:

$$\{a, b, c\}, \quad \{d, e, f\}.$$

The run outputs the original cut $\{a, b, c\} \mid \{d, e, f\}$, of size 3.

Another run can fail



If we ever contract one of the red edges, we merge vertices from opposite sides of this particular minimum cut. No later contraction can separate them again. The cut is lost.

Algorithm 1: Karger's contraction algorithm

Input: Connected undirected multigraph G, $n \geq 2$
Output: A cut of the original graph, and its number of crossing edges
$H \leftarrow G$; initially each current vertex represents one original vertex
while *H has more than two supervertices* **do**
    Sample an edge uniformly among all current edge *instances*
    Contract its endpoints (remove loops; keep parallel edges)
$S \leftarrow$ original vertices represented by either remaining supervertex
return $(S, |\delta_G(S)|)$

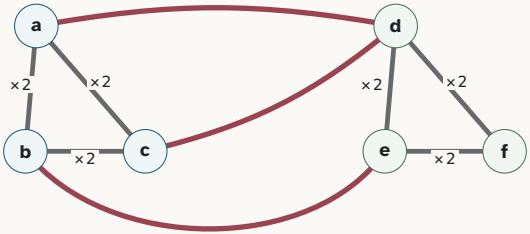
We will analyze the probability of returning a particular minimum cut.

PART 2

Why Karger’s algorithm works

2

Fix a particular minimum cut



Fix a minimum cut C, consisting of λ crossing edges.

$$\mathbb{P}(\text{output } C) = \mathbb{P}(\text{never contract an edge crossing } C).$$

This is because contractions only merge vertex sets: the two sides of C remain separable exactly when no crossing edge is contracted.

How likely is a dangerous contraction?

Suppose the fixed cut C has survived until k supervertices remain.

- Exactly λ remaining edges cross C.
- The random contraction chooses uniformly among all m_k remaining edge instances.

$$\mathbb{P}(\text{destroy } C \text{ next} \mid C \text{ survived}) = \frac{\lambda}{m_k}.$$

To control this probability, we need a *lower bound* on m_k.

Why does each supervertex have degree at least λ?

Take any current supervertex U. It represents a nonempty proper set S_U of original vertices. Every current edge incident to U comes from an original edge with exactly one endpoint in S_U.

$$\deg_H(U) = |\delta_G(S_U)| \geq \lambda.$$

The last inequality holds because λ is the size of the smallest cut in the *original* graph.

Self-loops are excluded from the degree; parallel edges count separately.

Count edges using the degree bound

With k supervertices, the sum of degrees counts each current edge twice:

$$2m_k = \sum_{U \in V(H)} \deg_H(U) \geq k\lambda.$$

Therefore

$$m_k \geq \frac{k\lambda}{2}, \qquad \mathbb{P}(\text{destroy } C \text{ next} \mid C \text{ survived}) \leq \frac{2}{k}.$$

So the next contraction preserves C with conditional probability at least $1 - 2/k$.

Multiply conditional survival probabilities

We contract from n supervertices down to 2. The chain rule gives

$$\mathbb{P}(C \text{ survives to the end}) \geq \prod_{k=3}^{n} \left(1 - \frac{2}{k}\right).$$

Now telescope:

$$\prod_{k=3}^{n} \frac{k-2}{k} = \frac{1 \cdot 2 \cdots (n-2)}{3 \cdot 4 \cdots n} = \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}.$$

Theorem. Any particular minimum cut is returned by one run with probability at least $1/\binom{n}{2}$.

Amplification by independent repetitions

One Karger run succeeds with probability at least $p = 1/\binom{n}{2}$. We can run it repeatedly.

Input: Graph G and number of repetitions $r \geq 1$
Output: Smallest cut found among r independent runs
$C_* \leftarrow \mathrm{Karger}(G)$
for $j \leftarrow 2$ **to** r **do**
    $C \leftarrow \mathrm{Karger}(G)$ using fresh random choices
    if $|C| < |C_*|$ **then**
        $C_* \leftarrow C$
return C_*

$$\Pr(\text{all runs miss a minimum cut}) \leq (1-p)^r \leq e^{-pr}.$$

Thus $r \geq \binom{n}{2} \ln(1/\delta)$ gives success probability at least $1 - \delta$.

PART 3

Karger–Stein: recursive contraction

3

Algorithm 2: Contract to t supervertices

Input: Current multigraph H with k supervertices, integer $2 \le t \le k$
Output: A new random graph with exactly t supervertices
$H' \leftarrow$ an independent copy of H
while *H' has more than t supervertices* **do**
    Sample a current edge instance uniformly at random
    Contract its endpoints in H'
return H'

ContractTo(H, t) stops *early*. Karger continues to two vertices. All current vertices retain a representation of which original vertices they contain.

How much contraction is safe?

Fix a minimum cut C of the input graph with n vertices. Stop contracting at t vertices rather than at 2. The same survival argument gives

$$\mathbb{P}(C \text{ survives to } t) \geq \prod_{k=t+1}^{n} \frac{k-2}{k} = \frac{t(t-1)}{n(n-1)}.$$

For $t \approx n/\sqrt{2}$, the lower bound is roughly $1/2$. We can remove many vertices while keeping a constant survival probability.

Choose a safe stopping size

Use the integer stopping rule

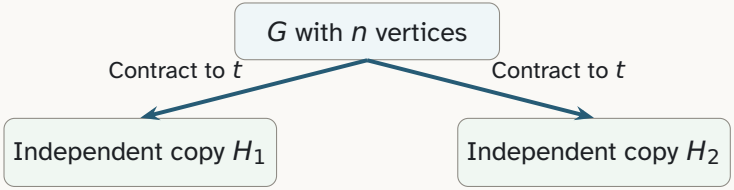
$$t(n) = \left\lceil \frac{n}{\sqrt{2}} + 1 \right\rceil.$$

Then $t(n)(t(n)-1) \geq n(n-1)/2$, and hence

$$\mathbb{P}(C \text{ survives to } t(n)) \geq \frac{1}{2}.$$

For $n > 6$, $t(n) < n$, so the recursive problem is smaller. At $n \leq 6$ we solve the minimum-cut problem exactly.

Why make two recursive branches?



Either branch may recover a minimum cut. We return whichever branch gives the smaller cut of the original graph.

Algorithm 3: Karger–Stein

Input: Connected contracted multigraph H with k supervertices
Output: A cut of the original graph, with its cut value
if $k \le 6$ **then**
  return *minimum cut of H found by enumerating its cuts*
$t \leftarrow \lceil k/\sqrt{2} + 1 \rceil$
$H_1 \leftarrow \mathrm{ContractTo}(H, t)$ using fresh randomness
$H_2 \leftarrow \mathrm{ContractTo}(H, t)$ independently
$C_1 \leftarrow \mathrm{KS}(H_1)$; $C_2 \leftarrow \mathrm{KS}(H_2)$
return *the smaller of C_1, C_2*

Each call returns a cut of the *original* graph. A surviving cut has the same crossing-edge count in every contracted graph.

One branch: two events must occur

Fix any minimum cut C. A sufficient condition for one branch to find a minimum cut is that

1. the contraction to t preserves C; and
2. the recursive call finds a minimum cut of that contracted graph.

The first event has probability at least $1/2$. If the first event happens, C remains a minimum cut of the contracted graph: any cut of that graph corresponds to a cut of the original graph. Define $p(n)$ as a lower bound, valid for every n-vertex input graph, on the probability that KS returns s*ome minimum cut*.

Two independent branches improve the chance

For one branch, conditioning on the contracted graph gives

$$\mathbb{P}(\text{branch returns a minimum cut}) \geq \tfrac{1}{2}p(t).$$

The branches use independent randomness. Their probability of both failing to find any minimum cut is at most

$$\left(1 - \tfrac{1}{2}p(t)\right)^2.$$

Thus, for $n > 6$,

$$\boxed{p(n) \geq 1 - \left(1 - \frac{p(t(n))}{2}\right)^2.}$$

The base case is exact: $p(n) = 1$ when $n \leq 6$.

A recurrence we can simplify

Expanding the square,

$$p(n) \geq p(t) - \frac{p(t)^2}{4} = p(t)\left(1 - \frac{p(t)}{4}\right).$$

It is awkward to iterate this inequality directly. Taking reciprocals turns the multiplicative loss into an additive one:

$$\frac{1}{p(n)} \leq \frac{1}{p(t)} + \frac{1}{4(1 - p(t)/4)} \leq \frac{1}{p(t)} + \frac{1}{3}.$$

The last step uses $0 < p(t) \leq 1$.

How fast does the recursive graph shrink?

For a graph with $k > 6$ supervertices, Karger–Stein recurses on graphs with

$$t(k) = \left\lceil k/\sqrt{2} + 1 \right\rceil$$

supervertices. Here are the sizes on one path when we start with 100:

$$100 \longrightarrow 72 \longrightarrow 52 \longrightarrow 38$$

How fast does the recursive graph shrink?

For a graph with $k > 6$ supervertices, Karger–Stein recurses on graphs with

$$t(k) = \left\lceil k/\sqrt{2} + 1 \right\rceil$$

supervertices. Here are the sizes on one path when we start with 100:

$$100 \longrightarrow 72 \longrightarrow 52 \longrightarrow 38 \longrightarrow 28 \longrightarrow 21 \longrightarrow 16$$

At each step the problem is smaller. To analyze the algorithm, we need a bound on the *number* of steps, not just that the process terminates.

How fast does the recursive graph shrink?

For a graph with $k > 6$ supervertices, Karger–Stein recurses on graphs with

$$t(k) = \left\lceil k/\sqrt{2} + 1 \right\rceil$$

supervertices. Here are the sizes on one path when we start with 100:

$$100 \longrightarrow 72 \longrightarrow 52 \longrightarrow 38 \longrightarrow 28 \longrightarrow 21 \longrightarrow 16$$

$$16 \longrightarrow 13 \longrightarrow 11 \longrightarrow 9 \longrightarrow 8 \longrightarrow 7 \longrightarrow 6.$$

At each step the problem is smaller. To analyze the algorithm, we need a bound on the *number* of steps, not just that the process terminates.

The recursion has logarithmic depth

The ceiling in the stopping rule gives

$$t(k) \le \frac{k}{\sqrt{2}} + 2.$$

The recursion has logarithmic depth

The ceiling in the stopping rule gives

$$t(k) \le \frac{k}{\sqrt{2}} + 2.$$

For $k \ge 16$, we have $2 \le k/8$, so

$$t(k) \le \left(\frac{1}{\sqrt{2}} + \frac{1}{8} \right) k < 0.84k.$$

The recursion has logarithmic depth

The ceiling in the stopping rule gives

$$t(k) \le \frac{k}{\sqrt{2}} + 2.$$

For $k \ge 16$, we have $2 \le k/8$, so

$$t(k) \le \left(\frac{1}{\sqrt{2}} + \frac{1}{8}\right) k < 0.84k.$$

Thus, until the size drops below 16, each step multiplies it by at most 0.84. After j steps, the size is at most $n(0.84)^j$.

$$n(0.84)^j < 16 \quad \text{once} \quad j = O(\log n).$$

For $7 \le k \le 15$, $t(k) < k$; at most nine more steps reach the exact base case $k \le 6$.

Finish the success-probability proof

Recall the bound derived from independent branches:

$$\frac{1}{p(k)} \le \frac{1}{p(t(k))} + \frac{1}{3}.$$

Starting from n, follow the sequence $n, t(n), t(t(n)), \ldots$ down to the base case. We showed that the size decreases geometrically above a fixed constant and then reaches the base case in a constant number of extra steps. Thus the path has $L = O(\log n)$ steps.

$$\frac{1}{p(n)} \le \underbrace{\frac{1}{p(n_L)}}_{=1} + \underbrace{\frac{1}{3} + \cdots + \frac{1}{3}}_{L \text{ terms}} = 1 + \frac{L}{3}.$$

Therefore $p(n) = \Omega(1/\log n)$.

PART 4

Efficient implementation and running time

4

Representing the current graph

A line labeled w represents w parallel edges. We store those counts in a symmetric array W:

$$W_{ij} = \text{number of edge instances joining supervertices } i \text{ and } j.$$

Representing the current graph

A line labeled w represents w parallel edges. We store those counts in a symmetric array W:

$$W_{ij} = \text{number of edge instances joining supervertices } i \text{ and } j.$$

At the root, allocate an $n \times n$ array. **Its size never changes during that contraction phase.** A recursive call works with a separate compact copy of its current graph. Maintain a list

$A = (a_1, \ldots, a_k)$ of the *active row indices* and an array d of current vertex degrees. Initially, $A = (1, \ldots, n)$.

Representing the current graph

A line labeled w represents w parallel edges. We store those counts in a symmetric array W:

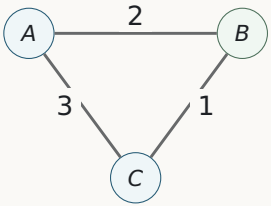
$$W_{ij} = \text{number of edge instances joining supervertices } i \text{ and } j.$$

At the root, allocate an $n \times n$ array. **Its size never changes during that contraction phase.** A recursive call works with a separate compact copy of its current graph. Maintain a list

$A = (a_1, \ldots, a_k)$ of the *active row indices* and an array d of current vertex degrees. Initially, $A = (1, \ldots, n)$. Only rows and columns whose indices are in A represent current supervertices.

A contraction deletes one entry from A, so k decreases by one; it does *not* resize W.

Edge multiplicities as matrix entries



$$W = \begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 2 & 3 \\ B & 2 & 0 & 1 \\ C & 3 & 1 & 0 \end{array}$$

Each line is an edge *bundle*; its label is the number of parallel edge instances. The matrix stores these counts, not geometric lengths or costs.

Matrix contraction: merge A and B

Before the merge,

$$W = \begin{pmatrix} 0 & 2 & 3 \\ 2 & 0 & 1 \\ 3 & 1 & 0 \end{pmatrix}.$$

To merge rows/columns A and B, add their connections to every *other* supervertex:

$$W_{AB,C} = W_{A,C} + W_{B,C} = 3 + 1 = 4.$$

Delete the A–B edges as self-loops, and remove one row and column:

$$W' = \begin{array}{c|cc} & AB & C \\ \hline AB & 0 & 4 \\ C & 4 & 0 \end{array}.$$

The active matrix shrinks from 3×3 to 2×2.

A contraction changes one row and one column

We contract active supervertex v into active supervertex u. For every other active index x:

$$W_{ux} \leftarrow W_{ux} + W_{vx}, \qquad W_{xu} \leftarrow W_{ux}.$$

A contraction changes one row and one column

We contract active supervertex v into active supervertex u. For every other active index x:

$$W_{ux} \leftarrow W_{ux} + W_{vx}, \qquad W_{xu} \leftarrow W_{ux}.$$

Edges between u and v become loops and are discarded. If $w = W_{uv}$ was their multiplicity, the degree and total-degree updates are

$$d_u \leftarrow d_u + d_v - 2w, \qquad D \leftarrow D - 2w.$$

A contraction changes one row and one column

We contract active supervertex v into active supervertex u. For every other active index x:

$$W_{ux} \leftarrow W_{ux} + W_{vx}, \qquad W_{xu} \leftarrow W_{ux}.$$

Edges between u and v become loops and are discarded. If $w = W_{uv}$ was their multiplicity, the degree and total-degree updates are

$$d_u \leftarrow d_u + d_v - 2w, \qquad D \leftarrow D - 2w.$$

Finally, delete v from the active-index list. Its old matrix row and column remain allocated but are ignored thereafter. No matrix row or column is moved.

Removing an active index: a complete example

Active indices before merging 2 into 1:



Replace the list entry 2 by its final entry 4:



The array W still has four allocated rows and columns. Rows 1, 3, and 4 represent the three current supervertices.

When sampling or updating, iterate through the entries of A, not through all n rows.

Removing an active index: a complete example

Active indices before merging 2 into 1:



Replace the list entry 2 by its final entry 4:



The active list changes from $[1, 2, 3, 4]$ to $[1, 4, 3]$ in $O(1)$ list operations. **No rows of W are swapped.**

When sampling or updating, iterate through the entries of A, not through all n rows.

Algorithm: contract an edge in the matrix

Input: Array W, degrees d, total degree D, active list A, positions $p \neq q$ in A
Output: Updated graph representation with one fewer active vertex
$u \leftarrow A[p]$; $v \leftarrow A[q]$; $w \leftarrow W_{uv}$
foreach $x \in A$ *with* $x \notin \{u, v\}$ **do**
  $W_{ux} \leftarrow W_{ux} + W_{vx}$; $W_{xu} \leftarrow W_{ux}$
$d_u \leftarrow d_u + d_v - 2w$; $D \leftarrow D - 2w$
$W_{uu} \leftarrow 0$
$A[q] \leftarrow A[k]$; shorten A from length k to $k-1$
return (W, d, D, A)

Cost: one scan over the k active indices, plus a constant number of assignments: $O(k)$ operations.

Weighted sampling: a simple primitive

To choose index i with probability proportional to w_i, let $Q = \sum_i w_i$ and draw a uniform integer $R \in \{1, \ldots, Q\}$.

$$\underbrace{1, 2, 3, 4, 5}_{A} \mid \underbrace{6, 7, 8}_{B} \mid \underbrace{9, 10, 11, 12}_{C}$$

Weighted sampling: a simple primitive

To choose index i with probability proportional to w_i, let $Q = \sum_i w_i$ and draw a uniform integer $R \in \{1, \dots, Q\}$.

$$\underbrace{1, 2, 3, 4, 5}_{A} \mid \underbrace{6, 7, 8}_{B} \mid \underbrace{9, 10, 11, 12}_{C}$$

Here the weights are $(5, 3, 4)$. If $R = 7$, choose B. Each index is chosen with probability equal to the length of its interval divided by 12.

Algorithm: sample an index by weight

Input: Indices $a_1, \ldots, a_k$ with nonnegative integer weights $w_1, \ldots, w_k$, $Q = \sum_i w_i > 0$
Output: Index a_i with probability w_i/Q
Draw R uniformly from $\{1, \ldots, Q\}$
$s \leftarrow 0$
for $i \leftarrow 1$ **to** k **do**
    $s \leftarrow s + w_i$
    if $s \geq R$ **then**
        return a_i

At most k weights are examined. The running time is $O(k)$.

Algorithm: sample an edge uniformly

Write $D = \sum_{u \in A} d_u$, twice the number of current edge instances.

Input: Current matrix W, active indices A, row sums d, total degree D
Output: Endpoints of a uniformly sampled edge instance
$u \leftarrow \mathrm{WeightedSample}(A, (d_x)_{x \in A})$
$v \leftarrow \mathrm{WeightedSample}(A, (W_{ux})_{x \in A})$
return (u, v)

Algorithm: sample an edge uniformly

Write $D = \sum_{u \in A} d_u$, twice the number of current edge instances.

Input: Current matrix W, active indices A, row sums d, total degree D
Output: Endpoints of a uniformly sampled edge instance
$u \leftarrow \mathrm{WeightedSample}(A, (d_x)_{x \in A})$
$v \leftarrow \mathrm{WeightedSample}(A, (W_{ux})_{x \in A})$
return (u, v)

First choose a supervertex proportionally to its degree; then choose a neighbor proportionally to the number of parallel edges connecting them. Both scans involve only the k active entries.

Numerical example of the sampler

With

$$W = \begin{pmatrix} 0 & 2 & 3 \\ 2 & 0 & 1 \\ 3 & 1 & 0 \end{pmatrix}, \qquad (d_A, d_B, d_C) = (5, 3, 4), \quad D = 12,$$

we obtain

$$\mathbb{P}(U = A) = \frac{5}{12}, \qquad \mathbb{P}(V = B \mid U = A) = \frac{2}{5}.$$

Therefore

$$\mathbb{P}(U = A, V = B) = \frac{5}{12}\frac{2}{5} = \frac{1}{6}.$$

The reverse orientation also has probability $1/6$, so

$$\mathbb{P}(\{U, V\} = \{A, B\}) = \frac{1}{3} = \frac{2}{6}.$$

Exactly two of the six edge instances connect A and B.

Why does the sampler choose edges uniformly?

Why does the sampler choose edges uniformly?

For any distinct i, j, the probability of the ordered pair is

$$\mathbb{P}(U = i, V = j) = \frac{d_i}{D}\frac{W_{ij}}{d_i} = \frac{W_{ij}}{D}.$$

Either orientation can be selected:

$$\mathbb{P}(\{U, V\} = \{i, j\}) = \frac{2W_{ij}}{D} = \frac{W_{ij}}{m}, \qquad m = D/2.$$

There are W_{ij} parallel edge instances between i, j, so this is exactly the probability of choosing one of those instances uniformly from all m edges.

Counting the work of one contraction

With k active supervertices, we have just analyzed these operations:

Operation	Cost
Choose first endpoint from degrees	$O(k)$
Choose second endpoint from its matrix row	$O(k)$
Add incident edge counts in the surviving row	$O(k)$
Update degrees and remove an active list entry	$O(1)$

Counting the work of one contraction

With k active supervertices, we have just analyzed these operations:

Operation	Cost
Choose first endpoint from degrees	$O(k)$
Choose second endpoint from its matrix row	$O(k)$
Add incident edge counts in the surviving row	$O(k)$
Update degrees and remove an active list entry	$O(1)$

Every contraction therefore takes $O(k)$ time. Contracting from n to t supervertices takes

$$\sum_{k=t+1}^{n} O(k) = O(n^2).$$

Recording the output cut

The matrix determines the edge multiplicities, but a cut must also identify which *original vertices* are on each side. This information is needed only when we output a cut.

Recording the output cut

The matrix determines the edge multiplicities, but a cut must also identify which *original vertices* are on each side. This information is needed only when we output a cut. Each time two supervertices merge, record that pair of merges in a separate history. When two supervertices remain, follow that history to identify the original vertices in each group. This uses constant additional bookkeeping per contraction; traversing the history to produce the final two vertex sets takes $O(n)$ time. The merge history is separate from the weighted adjacency matrix.

The cost of one Karger–Stein call

Consider a recursive call with $k > 6$ current supervertices. Read its pseudocode operation by operation:

Work done by this call	Time
Copy the $k \times k$ entries for the active vertices into two compact arrays	$O(k^2)$
Contract each copy down to $t(k)$ vertices	$O(k^2)$ total
Recursively solve both smaller graphs	$2T(t(k))$
Compare the two returned cut values	$O(1)$

Consequently,

$$\boxed{T(k) \le 2T(t(k)) + ck^2}, \quad t(k) = \left\lceil k/\sqrt{2} + 1 \right\rceil.$$

For $k \le 6$, enumerating the constant number of cuts costs $O(1)$.

Work per recursion level: first ignore rounding

Without rounding or the additive constant, sizes fall by $1/\sqrt{2}$ per level. At depth j:

$$\underbrace{2^j}_{\text{calls}} \qquad \underbrace{\frac{n}{(\sqrt{2})^j}}_{\text{size per call}}.$$

Each call spends time proportional to the square of its size, so

$$2^j \cdot c\left(\frac{n}{(\sqrt{2})^j}\right)^2 = cn^2.$$

The branching factor 2 is exactly canceled by the squared shrinkage factor $1/2$. The actual stopping rule includes rounding and $+1$. We account for those on the next slide, rather than treating the approximation as exact.

Rounding: bounding sizes at depth j

The exact rule obeys $k_{j+1} \leq ak_j + 2$, where $a = 1/\sqrt{2}$.

Rounding: bounding sizes at depth j

The exact rule obeys $k_{j+1} \le a k_j + 2$, where $a = 1/\sqrt{2}$. Apply this repeatedly:

$$k_j \le a^j n + 2(1 + a + \cdots + a^{j-1}).$$

Rounding: bounding sizes at depth j

The exact rule obeys $k_{j+1} \le \alpha k_j + 2$, where $\alpha = 1/\sqrt{2}$. Apply this repeatedly:

$$k_j \le \alpha^j n + 2(1 + \alpha + \cdots + \alpha^{j-1}).$$

Because the geometric sum is bounded by $1/(1-\alpha)$,

$$\boxed{k_j \le \alpha^j n + b}, \qquad b = \frac{2}{1-\alpha}.$$

This is the usual geometric shrinkage plus a fixed constant that accounts for rounding.

Rounding: work at one depth

At depth j, there are at most 2^j recursive calls, each of size at most $\alpha^j n + b$. Their nonrecursive work is bounded by

$$c2^j(\alpha^j n + b)^2.$$

Rounding: work at one depth

At depth j, there are at most 2^j recursive calls, each of size at most $a^j n + b$. Their nonrecursive work is bounded by

$$c2^j(a^j n + b)^2.$$

Using $(x + y)^2 \leq 2x^2 + 2y^2$ and $2a^2 = 1$ gives

$$\begin{aligned} c2^j(a^j n + b)^2 &\leq 2cn^2(2a^2)^j + 2cb^2 2^j \\ &= 2cn^2 + 2cb^2 2^j. \end{aligned}$$

Rounding: work at one depth

At depth j, there are at most 2^j recursive calls, each of size at most $a^j n + b$. Their nonrecursive work is bounded by

$$c2^j(a^j n + b)^2.$$

Using $(x + y)^2 \le 2x^2 + 2y^2$ and $2a^2 = 1$ gives

$$\begin{aligned} c2^j(a^j n + b)^2 &\le 2cn^2(2a^2)^j + 2cb^2 2^j \\ &= 2cn^2 + 2cb^2 2^j. \end{aligned}$$

At depths where $a^j n$ is bounded below by a fixed constant, $2^j = O(n^2)$. After that there are only $O(1)$ more levels before the base case, by the depth argument in Part 3. Thus every level costs $O(n^2)$.

Summing the work of Karger–Stein

We established two facts:

1. A recursive call on k vertices spends $O(k^2)$ outside its recursive calls.
2. The recursion has $O(\log n)$ levels, with $O(n^2)$ total work per level.

Therefore one invocation costs

$$\boxed{T(n) = O(n^2 \log n).}$$

If the input is an edge list containing m edge instances, constructing the root matrix requires another $O(n^2 + m)$ operations. **This is the cost of one invocation, not the cost of amplification.** To reduce failure probability, we must repeat independent invocations.

Amplifying Karger–Stein

One invocation finds a minimum cut with probability at least $c/\log n$ for some constant $c > 0$. Run it independently r times and return the smallest cut found:

$$\mathbb{P}(\text{all fail}) \le \left(1 - \frac{c}{\log n}\right)^r \le e^{-cr/\log n}.$$

Taking $r = O(\log n \log(1/\delta))$ gives failure probability at most δ. Using $O(n^2 \log n)$ time per invocation, this yields

$$O\big(n^2 \log^2 n \log(1/\delta)\big)$$

under the matrix implementation.

PART 5

A counting consequence

5

How many minimum cuts can there be?

Suppose G has N distinct minimum cuts $C_1, \ldots, C_N$. A single Karger run outputs exactly one cut, so the events "output C_i" are disjoint:

$$1 \geq \sum_{i=1}^{N} \mathbb{P}(\text{output } C_i).$$

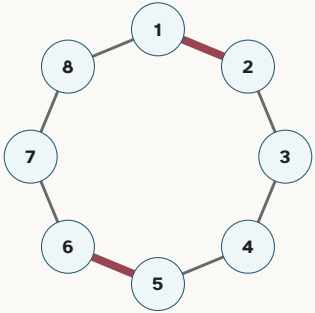
But the success theorem gives

$$\mathbb{P}(\text{output } C_i) \geq \frac{1}{\binom{n}{2}} \quad \text{for every } i.$$

Therefore

$$\boxed{N \leq \binom{n}{2}}.$$

The bound is sharp: an n-cycle



Removing any two different edges of a cycle disconnects it.

Different pairs define different cuts.

$$\text{Number of minimum cuts} = \binom{n}{2}.$$

Questions?

Technical appendix

Appendix: exact matrix and degree update

For a contraction of distinct active vertices u, v with $w = W_{uv}$, define for all $x \notin \{u, v\}$

$$W'_{ux} = W_{ux} + W_{vx}, \qquad d'_u = d_u + d_v - 2w.$$

Other surviving degrees d_x do not change; d_v is deleted. Set $W'_{uu} = 0$ and remove the active row/column v. The matrix can be stored in an $n \times n$ buffer; swap v with the last active index to avoid physically shifting rows. A full row/column swap requires $O(k)$ changes. Maintain $D = \sum d_i = 2m$, updating $D \leftarrow D - 2w$.

Appendix: why not scan the whole matrix?

Scanning all k^2 matrix entries to choose each random edge costs $O(k^2)$ per contraction and $O(n^3)$ for a full contraction phase. Instead, maintain the row sums d_i. A linear scan selects U with probability d_i/D, and a scan of row U selects V with probability W_{UV}/d_U. Since each of these scans uses $O(k)$ operations, sampling an edge and merging its endpoints both fit within $O(k)$ time.