Lecture 16
Min Cut and Karger’s Algorithm
Announcements

• HW 7 due Friday
• HW 8 released Friday
  • Psych! **There is no HW8.**
  • Min-cut and max-flow will be on the **final exam** the same way BFS/DFS/SCCs were on the midterm.
  • If you want practice, we will suggest some problems.

• **FINAL EXAM:**
  • Friday June 9
  • 3:30 – 6:30pm
  • Hewlett 200
  • **THE FINAL WILL BE HARDER THAN THE MIDTERM.**
Motwani Distinguished Lecture tomorrow.

- Thursday June 1, 4:15 pm, Encina Hall
- Piotr Indyk will talk about:
  
  **Beyond P vs. NP:**

  Quadratic-Time Hardness For Big Data Problems

- You guys keep asking me if we can do better than the $O(n^2)$ algorithms that we’ve seen in this class.
  
  - “That’s an open problem,” blah blah blah

- This talk will give you some answers!

http://theory.stanford.edu/motwani_lecture/
Last time

• Minimum Spanning Trees!
  • Prim’s Algorithm
  • Kruskal’s Algorithm
Today

• Minimum Cuts!
  • Karger’s algorithm
  • Karger-Stein algorithm

• Back to randomized algorithms
  • but in a different way than we’ve seen so far
Recall: cuts in graphs

• A cut is a partition of the vertices into two nonempty parts.

*For today, all graphs are undirected and unweighted.
Recall: cuts in graphs

- A cut is a partition of the vertices into two nonempty parts.

*For today, all graphs are undirected and unweighted.
This is not a cut
This is a cut
This is a cut

These edges **cross the cut**.
- They go from one part to the other.
A (global) minimum cut is a cut that has the fewest edges possible crossing it.
A (global) minimum cut is a cut that has the fewest edges possible crossing it.
Why “global”?

• Next week we’ll talk about **min s-t cuts**

• Today, there are no special vertices, so the minimum cut is “global.”
A (global) minimum cut is a cut that has the fewest edges possible crossing it.
Why might we care about global minimum cuts?

• One example is image segmentation:
Why might we care about global minimum cuts?

• One example is image segmentation:

• We’ll see more applications for other sorts of min-cuts next week
Karger’s algorithm

• Finds **global minimum cuts** in undirected graphs

• Randomized algorithm
  • But a different sort of randomized algorithm than QuickSort!

• Karger’s algorithm **might be wrong**.
  • While QuickSort just might be slow.

• Why would we want an algorithm that might be wrong?
  • **With high probability it won’t be wrong.**
  • Maybe the stakes are low and the cost of a deterministic algorithm is high.
Different sorts of gambling

- QuickSort is a **Las Vegas randomized algorithm**
  - It is always correct.
  - It might be slow.

Formally:
- For all inputs A, QuickSort(A) returns a sorted array.
- For all inputs A, with high probability over the choice of pivots, QuickSort(A) runs quickly.

Yes, this is a technical term.
Different sorts of gambling

• Karger’s Algorithm is a Monte Carlo randomized algorithm
  • It is always fast.
  • It might be wrong.

Formally:
• For all inputs G, with probability at least ___ over the randomness in Karger’s algorithm, Karger(G) returns a minimum cut.

• For all inputs G, with probability 1 Karger’s algorithm runs in time no more than ____.

Algorithms that might be slow and might also be wrong are called “Atlantic City” algorithms.
Karger’s Algorithm

• Pick a random edge.
• **Contract** it.
• Repeat until you only have two vertices left.

Why is this a good idea? We’ll see shortly.
Karger’s algorithm
Karger’s algorithm

random edge!
Karger’s algorithm

Create a supernode!

Create a superedge!

Create a superedge!
Karger’s algorithm

Create a supernode!

Create a superedge!

Create a superedge!
Karger’s algorithm
Karger’s algorithm

Create a supernode!

Create a superedge!
Karger’s algorithm

Create a supernode!

Create a superedge!

Create a superedge!
Karger’s algorithm

random edge!
Karger’s algorithm

The diagram shows a graph with vertices labeled as follows:
- a, b
- c, d
- f
- g

The edges and their corresponding sets are:
- {a, b} with sets: {c, a}, {c, b}, {d, a}, {d, b}
- {e, d}
- {f, e}, {f, h}
- {g, e}, {g, h}
Karger’s algorithm

random edge!
Karger’s algorithm

a,b,c,d
{e,b} {e,d}
e,h
{f,e} {f,h}
g
{g,e} {g,h}
Karger’s algorithm
Karger’s algorithm
Karger’s algorithm

Graph with vertices labeled as:
- a, b, c, d
- e, h, g

Edges:
- e, b
- e, d
- f, g
- f, e
- f, h

Random edge!
Karger’s algorithm

\( \{e, b\} \{e, d\} \)
Karger’s algorithm

Now stop!
- There are only two nodes left.

The minimum cut is given by the remaining super-nodes:
- \{a,b,c,d\} and \{e,h,f,g\}
- Cut across \{e,b\}, \{e,d\}
Karger’s algorithm

The minimum cut is given by the remaining super-nodes:
• \{a,b,c,d\} and \{e,h,f,g\}
• Cut across \{e,b\}, \{e,d\}
# Pseudocode

Let $\bar{u}$ denote the SuperNode in $\Gamma$ containing $u$.

Let $E_{\bar{u},\bar{v}}$ be the SuperEdge between $\bar{u}$ and $\bar{v}$.

## Karger($G=(V,E)$):

- $\Gamma = \{\text{SuperNode}(v) : v \in V\}$  
  // one supernode for each vertex
- $E_{\bar{u},\bar{v}} = \{(u,v)\}$ for $(u,v) \in E$  
  // one superedge for each edge
- $E_{\bar{u},\bar{v}} = \{}$ for $(u,v)$ not in $E$.  
- $F = \{(u,v)\} : (u,v) \in E$  
  // we’ll choose randomly from $F$

## while $|\Gamma| > 2$:

- $\{(u,v)\} \leftarrow$ uniformly random edge in $F$
- merge($u, v$)  
  // merge the SuperNode containing $u$ with the SuperNode containing $v$.  
- $F \leftarrow F \setminus E_{\bar{u},\bar{v}}$  
  // remove all the edges in the SuperEdge between those SuperNodes.

## return the cut given by the remaining two superNodes.

## merge($u, v$):  

- $\bar{x} = \text{SuperNode}(\bar{u} \cup \bar{v})$  
  // merge also knows about $\Gamma$ and the $E_{\bar{u},\bar{v}}$’s
- for each $w$ in $\Gamma \setminus \{\bar{u}, \bar{v}\}$:
  - $E_{\bar{x},\bar{w}} = E_{\bar{u},\bar{w}} \cup E_{\bar{v},\bar{w}}$
  - Remove $\bar{u}$ and $\bar{v}$ from $\Gamma$ and add $\bar{x}$.

**total runtime $O(n^2)$**

We can do a bit better with fancy data structures, but let’s go with this for now.
Why did that work?

• We got really lucky!
• This could have gone wrong in so many ways.
Karger’s algorithm

Say we had chosen this edge
Karger’s algorithm

Say we had chosen this edge

Now there is no way we could return a cut that separates b and e.
Even worse

If the algorithm *EVER* chooses either of these edges, it will be wrong.
How likely is that?

• For this particular graph, I did it 10,000 times:

Frequency of cut sizes that this algorithm returns

- Cut size >5
- Cut size 5
- Cut size 4
- Cut size 3
- Cut size 2
- Cut size 1

The algorithm is not correct very often! 372 out of 10,000 trials, or about 3.7% of the time...
But this is better than it could be

- Suppose that we chose cuts \textbf{uniformly at random}.
  - That is, pick a random way to split the vertices into 2 parts.
But this is better than it could be

- Suppose that we chose cuts uniformly at random.
  - That is, pick a random way to split the vertices into 2 parts.

- The probability of choosing the minimum cut is*...

\[
\frac{\text{number of min cuts in that graph}}{\text{number of ways to split 8 vertices in 2 parts}} = \frac{2}{2^8 - 2}
\]

- After 10,000 trials, I’d expect to get the min cut \(\frac{2 \cdot 10,000}{2^8 - 2} \approx 80\) times.

- But I actually got the right answer **over 350 times**.

- This is **better than completely random!**
What’s going on?

• Which is more likely?

A: The algorithm never chooses either of the edges in the minimum cut.

B: The algorithm never chooses any of the edges in this big cut.

• Neither A nor B are very likely, but A is more likely than B.

Lucky the lackadaisical lemur

It’s unlikely that we’ll hit the min cut since it’s so small!
Why does this help?

- We are still wrong 96.3% of the time.
- The idea: **repeat**!
  - If I am correct about 3.7% of the time...
  - Then do it enough times (at least $1/0.037 = 27$ times)
    - take the best cut ever returned.

**Whoa there!** Where did this 3.7% number come from? We did a few experiments on just one graph. What is the right answer for a general graph?
Plucky is right

• We need to figure out exactly what the probability of success is with this algorithm.

• It’s going to be small, but how small?
  • If it’s **not too small**, then we can just repeat a few times.
  • If it is **too small**, then “a few” might be “lots.”
Claim

The probability that Karger’s algorithm returns a minimum cut is at least \( \frac{1}{\binom{n}{2}} \).

In this case, \( \frac{1}{\binom{8}{2}} = 0.036 \), so at least 3.6% of the time. That’s about what we saw empirically.
Before we prove the claim

• Why is it helpful?

Again I ran Karger’s algorithm repeatedly on this graph:

The failure probability quickly gets really really small. But how quickly?
A computation

• Suppose:
  • the probability of successfully returning a minimum cut is \( p \in [0, 1] \),
  • we want failure probability at most \( \delta \in (0,1) \).

• \( \Pr[ \text{don’t return a min cut in } T \text{ trials } ] = (1 - p)^T \)

• So if \( p = 1/(\binom{n}{2}) \), and choose \( T = \binom{n}{2} \ln(1/\delta) \)

• \( \Pr[ \text{don’t return a min cut in } T \text{ trials } ] \)
  • \( = (1 - p)^T \)
  • \( \leq (e^{-p})^T \)
  • \( = e^{-pT} \)
  • \( = e^{-\ln(\frac{1}{\delta})} \)
  • \( = \delta \)

If we repeat \( T = \binom{n}{2} \ln(1/\delta) \) times, we win with probability at least \( 1 - \delta \).
Theorem
Assuming the claim about \( \frac{1}{C(n)} \) ...

• Suppose \( G \) has \( n \) vertices.

• Consider the following algorithm:
  • \( \text{bestCut} = \text{None} \)
  • \textbf{for} \( t = 1, \ldots, \binom{n}{2} \ln \left( \frac{1}{\delta} \right) \) :
    • \( \text{candidateCut} \leftarrow \text{Karger}(G) \) \hspace{1cm} \text{\( \backslash \) independent randomness}
    • \textbf{if} \( \text{candidateCut} \) is smaller than \( \text{bestCut} \):
      • \( \text{bestCut} \leftarrow \text{candidateCut} \)
  • \textbf{return} \( \text{bestCut} \)

• Then

\[
\Pr[ \text{this doesn't return a min cut} ] \leq \delta.
\]
What’s the running time?

• **Depends** on how we implement Karger’s algorithm.

• As stated, \( O \left( n^2 \cdot \binom{n}{2} \ln \left( \frac{1}{\delta} \right) \right) = O(n^4) \)

• If we use **union-find data structures**, we can do better.

These are the things we used to implement Kruskal’s algorithm last week.

Write pseudocode for a fast version of Karger’s algorithm! How fast can you make the asymptotic running time?

Let’s go with \( O(n^4) \) for now.

*Ollie the over-achieving ostrich*
Now let’s prove that claim
Claim

The probability that Karger’s algorithm returns a minimum cut is at least \( \frac{1}{\binom{n}{2}} \)
Now let’s prove that claim
Say that $S^*$ is a minimum cut.

• Suppose the edges that we choose are $e_1, e_2, ..., e_{n-2}$

• $\text{PR}[ \text{return } S^* ] = \text{PR}[ \text{none of the } e_i \text{ cross } S^* ]$

  $= \text{PR}[ e_1 \text{ doesn’t cross } S^* ]$

  $\times \text{PR}[ e_2 \text{ doesn’t cross } S^* \mid e_1 \text{ doesn’t cross } S^* ]$

  $\times \text{PR}[ e_{n-2} \text{ doesn’t cross } S^* \mid e_1,...,e_{n-3} \text{ don’t cross } S^* ]$
Focus in on: \( \text{PR}[ e_j \text{ doesn’t cross } S^* \mid e_1, \ldots, e_{j-1} \text{ don’t cross } S^* ] \)

- Suppose: After \( j-1 \) iterations, we haven’t messed up yet!
- What’s the probability of messing up now?

These two edges haven’t been chosen for contraction!
Focus in on: \( \mathbb{P}[\text{e}_j \text{ doesn’t cross } S^* \mid \text{e}_1, \ldots, \text{e}_{j-1} \text{ don’t cross } S^* ] \)

• Suppose: After j-1 iterations, we haven’t messed up yet!

• What’s the probability of messing up now?

• Say there are \( k \) edges that cross \( S^* \)

• Every remaining node has degree at least \( k \)
  • Otherwise we’d have a smaller cut.

• Thus, there are at least \( (n-j+1)k/2 \) edges total.
  • \( n - j + 1 \) nodes left, each with degree at least \( k \).

So the probability that we choose one of the \( k \) edges crossing \( S^* \) at step \( j \) is at most:

\[
\frac{k}{\frac{(n-j+1)k}{2}} = \frac{2}{n-j+1}
\]
Now let’s prove that claim
Say that $S^*$ is a minimum cut.

• Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$

• $\text{PR}[\text{return } S^* ] = \text{PR}[ \text{none of the } e_i \text{ cross } S^* ]$

  $= \text{PR}[ e_1 \text{ doesn’t cross } S^* ]$

  $\times \text{PR}[ e_2 \text{ doesn’t cross } S^* \mid e_1 \text{ doesn’t cross } S^* ]$

  $\ldots$

  $\times \text{PR}[ e_{n-2} \text{ doesn’t cross } S^* \mid e_1,\ldots,e_{n-3} \text{ don’t cross } S^* ]$
Now let’s prove that claim
Say that $S^*$ is a minimum cut.

• Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$
• $\text{PR}[\text{return } S^* ] = \text{PR}[\text{none of the } e_i \text{ cross } S^* ]$

$$= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \ldots \left(\frac{4}{6}\right) \left(\frac{3}{5}\right) \left(\frac{2}{4}\right) \left(\frac{1}{3}\right)$$
Now let’s prove that claim

Say that $S^*$ is a minimum cut.

• Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$

• $\text{PR}[$ return $S^*$ $] = \text{PR}[$ none of the $e_i$ cross $S^*$ $]$

$\begin{align*}
&= \left( \frac{n-2}{n} \right) \left( \frac{n-3}{n-1} \right) \left( \frac{n-4}{n-2} \right) \left( \frac{n-5}{n-3} \right) \left( \frac{n-6}{n-4} \right) \ldots \left( \frac{4}{n} \right) \left( \frac{3}{5} \right) \left( \frac{2}{4} \right) \left( \frac{1}{3} \right) \\
&= \left( \frac{2}{n(n-1)} \right) \\
&= \frac{1}{\binom{n}{2}}
\end{align*}$

CLAIM PROVED
Theorem
Assuming the claim about $1/\binom{n}{2}$ ...

• Suppose $G$ has $n$ vertices.
• Consider the following algorithm:
  • $\text{bestCut} = \text{None}$
  • $\textbf{for } t = 1, \ldots, \frac{n}{2}\ln \left(\frac{1}{\delta}\right) :$
    • $\text{candidateCut} \leftarrow \text{Karger}(G)$
    • $\textbf{if } \text{candidateCut} \text{ is smaller than } \text{bestCut}$:
      • $\text{bestCut} \leftarrow \text{candidateCut}$
  • $\textbf{return } \text{bestCut}$

Then
$$\Pr[ \text{this doesn’t return a min cut } ] \leq \delta.$$
What have we learned?

• If we randomly contract edges:
  • It’s unlikely that we’ll end up with a min cut.
  • But it’s not TOO unlikely
  • By repeating, we likely will find a min cut.

• Repeating this process:
  • Finds a global min cut in time $O(n^4)$, with probability 0.99.
  • We can run a bit faster if we use a union-find data structure.

*Note, in the lecture notes, we take $\delta = \frac{1}{n}$, which makes the running time $O(n^4 \log(n))$. It depends on how sure you want to be!
More generally

• Whenever we have a Monte-Carlo algorithm with a small success probability, we can **boost** the success probability by repeating it a bunch and taking the best solution.
Can we do better?

• Repeating $O(n^2)$ times is pretty expensive.
  • $O(n^4)$ total runtime to get success probability 0.99.

• The **Karger-Stein Algorithm** will do better!
  • The trick is that we’ll do the repetitions in a clever way.
  • $O(n^2 \log^2(n))$ runtime for the same success probability.

To see how we might save on repetitions, let’s run through Karger’s algorithm again.
Karger’s algorithm
Karger’s algorithm

There are 14 edges, 12 of which are good to contract.

Probability that we didn’t mess up: \( \frac{12}{14} \)
Karger’s algorithm

Create a supernode!

Create a superedge!

Create a superedge!
Karger’s algorithm

Create a supernode!

Create a superedge!
Karger’s algorithm

Probability that we didn’t mess up: \( \frac{11}{13} \)

Now there are only 13 edges, since the edge between a and b disappeared.
Karger’s algorithm

Create a supernode!

Create a superedge!
Karger’s algorithm

Create a supernode!

Create a superedge!

Create a superedge!
Karger’s algorithm

Probability that we didn’t mess up: \[ \frac{10}{12} \]

Now there are only 12 edges, since the edge between e and h disappeared.
Karger’s algorithm

\[ \text{Karger's algorithm} \]

Graph:
- Nodes: a, b, c, d, e, h, f, g
- Edges:
  - a, b to c, d (green, single edge)
  - c, d to e, h (green, single edge)
  - e, h to f, g (blue, single edge)
- Connections:
  - \{c, a\}, \{c, b\}, \{d, a\}, \{d, b\}
  - \{e, b\}, \{e, d\}
  - \{f, e\}, \{f, h\}, \{g, e\}, \{g, h\}
Karger’s algorithm

Probability that we didn’t mess up: $\frac{9}{11}$

random edge! (We pick at random from the original edges).

Karger’s algorithm
Karger’s algorithm

Probability that we didn’t mess up: 5/7
Karger’s algorithm

Vertices: a, b, c, d, e, h, g

Edges: 
- {a, b} {b, e}
- {e, d}
- {f, g} {f, e} {f, h}
Karger’s algorithm

Probability that we didn’t mess up: 3/5
Karger’s algorithm

\[\{e,b\} \{e,d\}\]
Karger’s algorithm

Now stop!

• There are only two nodes left.
Probability of not messing up

- At the beginning, it’s pretty likely we’ll be fine.
- The probability that we mess up gets worse and worse over time.

**Moral:** Repeating at the beginning is **wasteful**! Instead we should wait until our probability of success is a bit smaller.
Instead...

This branch made a bad choice.

But it’s okay since this branch made a good choice.
In words

• Run Karger’s algorithm on G for a bit.
  • Until there are $\frac{n}{\sqrt{2}}$ supernodes left.

• Then split into two independent copies, $G_1$ and $G_2$

• Run Karger’s algorithm on each of those for a bit.
  • Until there are $\frac{(n/\sqrt{2})}{\sqrt{2}} = \frac{n}{2}$ supernodes left in each.

• Then split each of those into two independent copies...

Why $\frac{n}{\sqrt{2}}$? We’ll see later.
In pseudocode

• **KargerStein(G = (V,E))**:  
  • n ← |V|  
  • if n < 4:  
    • find a min-cut by brute force \time O(1)  
  • Run Karger’s algorithm on G with independent repetitions until \[ \left\lfloor \frac{n}{\sqrt{2}} \right\rfloor \] nodes remain.  
  • \(G_1, G_2 \leftarrow \) copies of what’s left of G  
  • \(S_1 = \text{KargerStein}(G_1)\)  
  • \(S_2 = \text{KargerStein}(G_2)\)  
  • return whichever of \(S_1, S_2\) is the smaller cut.
Recursion tree

- $n$ nodes
  - $\frac{n}{\sqrt{2}}$ nodes
    - $\frac{n}{\sqrt{4}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
    - $\frac{n}{\sqrt{8}}$ nodes
  - $\frac{n}{\sqrt{4}}$ nodes
    - $\frac{n}{\sqrt{4}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
    - $\frac{n}{\sqrt{8}}$ nodes
  - $\frac{n}{\sqrt{4}}$ nodes
    - $\frac{n}{\sqrt{4}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
      - $\frac{n}{\sqrt{8}}$ nodes
    - $\frac{n}{\sqrt{8}}$ nodes

Make 2 copies

Contract a bunch of edges
Recursion tree

- depth is $\log_{\sqrt{2}}(n) = \frac{\log(n)}{\log(\sqrt{2})} = 2\log(n)$
- number of leaves is $2^{2\log(n)} = n^2$
Two questions

• Does this work?
• What’s the runtime?
At the $j$th level

- The amount of work per level is the amount of work needed to reduce the number of nodes by a factor of $\sqrt{2}$.

- That’s at most $O(n^2)$.
  - since that’s the time it takes to run Karger’s algorithm once, cutting down the number of supernodes to two.

- Our recurrence relation is...
  
  $$T(n) = 2T(n/\sqrt{2}) + O(n^2)$$

- The Master Theorem says...
  
  $$T(n) = O(n^2 \log(n))$$

Jedi Master Yoda.
Two questions

• Does this work?
• What’s the runtime?
First

Why $n/\sqrt{2}$?

• Suppose the first $n-t$ edges that we choose are $e_1, e_2, \ldots, e_{n-t}$

• $\text{PR}[ \text{none of the } e_i \text{ cross } S^* \text{ (up to the } n-t'\text{th}) ]$
  
  $\times \text{PR}[ e_2 \text{ doesn’t cross } S^* \mid e_1 \text{ doesn’t cross } S^* ]$

  $\times \text{PR}[ e_{n-t} \text{ doesn’t cross } S^* \mid e_1, \ldots, e_{n-t-1} \text{ don’t cross } S^* ]$

Suppose we contract $n-t$ edges, until there are $t$ supernodes remaining.
First

Why $n/\sqrt{2}$ ?

• Suppose the first $n-t$ edges that we choose are $e_1, e_2, \ldots, e_{n-t}$

• $\Pr[\text{none of the } e_i \text{ cross } S^* \text{ (up to the } n-t \text{'th})]$

  \[
  = \frac{(n-2)}{n} \frac{(n-3)}{n-1} \frac{(n-4)}{n-2} \frac{(n-5)}{n-3} \frac{(n-6)}{n-4} \cdots \frac{(t+1)}{t+3} \frac{t}{t+2} \frac{t-1}{t+1}
  \]

  \[
  = \frac{t \cdot (t-1)}{n \cdot (n-1)}
  \]

  Choose $t = n/\sqrt{2}$

\[
= \frac{n \sqrt{2} \cdot (\frac{n}{\sqrt{2}} - 1)}{n \cdot (n-1)} \approx \frac{1}{2}
\]

when $n$ is large

Suppose we contract $n-t$ edges, until there are $t$ supernodes remaining.
Recursion tree

- n nodes
  - $\frac{n}{\sqrt{2}}$ nodes
    - Contract a bunch of edges
      - $\frac{n}{\sqrt{4}}$ nodes
        - Make 2 copies
          - $\frac{n}{\sqrt{8}}$ nodes
            - Pr[ failure ] = 1/2
  - $\frac{n}{\sqrt{2}}$ nodes
    - Contract a bunch of edges
      - $\frac{n}{\sqrt{4}}$ nodes
        - Make 2 copies
          - $\frac{n}{\sqrt{8}}$ nodes
            - Pr[ failure ] = 1/2

- $\frac{n}{2\sqrt{2}}$ nodes

Pr[ failure ] = 1/2

etc.
Probability of success

Is a probability that there’s a path from the root to a leaf with no failures.

\[
\frac{n}{2\sqrt{2}} \text{ nodes}
\]

Each with probability 1/2
Analysis

- Say the tree has height $d$.
- Let $p_d$ be the probability that there’s a path from the root to a leaf that **doesn’t fail**.
  
  \[
  p_d = \frac{1}{2} \cdot \Pr \left[ \begin{array}{c}
  \text{Pr [wins]} \\
  \text{Pr [both win]}
  \end{array} \right]
  \]

  \[
  = \frac{1}{2} \cdot \left( p_{d-1} + p_{d-1} - p_{d-1}^2 \right)
  \]

  \[
  = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2
  \]
Recurrence relation

• **Claim**: for all $d$, $p_d \geq \frac{1}{d+1}$

• **Proof**: induction on $d$.
  
  • **Base case**: $1 \geq 1$. **YEP.**
  
  • **Inductive step**: say $d > 0$.
    
    • Suppose that $p_{d-1} \geq \frac{1}{d}$.
    
    • $p_d = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$
    
    • $\geq \frac{1}{d} - \frac{1}{2} \cdot \frac{1}{d^2}$
    
    • $\geq \frac{1}{d} - \frac{1}{d(d+1)}$
    
    • $= \frac{1}{d+1}$

• $p_d = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$

• $p_0 = 1$

We will probably blow by these computations quickly in lecture. Check them on your own!
What does that mean for Karger-Stein?

Claim: for all $d$, $p_d \geq \frac{1}{d+1}$

- For $d = 2\log(n)$
  - that is, $d$ = the height of the tree:

$$p_{2\log(n)} \geq \frac{1}{2\log(n) + 1}$$

- aka,

$$\Pr[\text{Karger-Stein is successful}] = \Omega \left( \frac{1}{\log(n)} \right)$$
Altogether now

• We can do the **same trick** as before to amplify the success probability.
  
  • Run Karger-Stein $O \left( \log(n) \cdot \log \left( \frac{1}{\delta} \right) \right)$ times
  
  • to achieve success probability $1 - \delta$.

• Choosing $\delta = 0.01$ as before, the total runtime is

\[
O(n^2 \log(n) \cdot \log(n)) = O(n^2 \log(n)^2)
\]

• That’s what we claimed!
What have we learned?

• Just repeating Karger’s algorithm isn’t the best use of repetition.
  • We’re probably going to be correct near the beginning.

• Instead, Karger-Stein repeats when it counts.
  • If we wait until there are \( \frac{n}{\sqrt{2}} \) nodes left, the probability that we fail is close to \( \frac{1}{2} \).

• This lets us find a global minimum cut in an undirected graph in time \( O(n^2 \log^2(n)) \).
  • Notice that we can’t do better than \( n^2 \) in a dense graph (we need to look at all the edges), so this is pretty good.
Recap

• Some algorithms:
  • Karger’s algorithm for global min-cut
  • Improvement: Karger-Stein

• Some concepts:
  • Monte Carlo algorithms:
    • Might be wrong, are always fast.
  • We can boost their success probability with repetition.
  • Sometimes we can do this repetition very cleverly.
Next time

• Another sort of min-cut:
  • s-t min-cut
  • also max-flow!