Randomized Algorithms
Part Three
Announcements

• Problem Set Three due on Monday (or Wednesday using a late period.)
• Problem Set Two graded; will be returned at the end of lecture.
Outline for Today

- **Global Minimum Cut**
  - What is the easiest way to split a graph into pieces?

- **Karger's Algorithm**
  - A simple randomized algorithm for finding global minimum cuts.

- **The Karger-Stein Algorithm**
  - A fast, simple, and elegant randomized divide-and-conquer algorithm.
Recap: Global Cuts
Disconnecting a Graph
Global Min Cuts

- A **cut** in a graph \( G = (V, E) \) is a way of partitioning \( V \) into two sets \( S \) and \( V - S \). We denote a cut as the pair \( (S, V - S) \).

- The **size** of a cut is the number of edges with one endpoint in \( S \) and one endpoint in \( V - S \). These edges are said to **cross** the cut.

- A **global minimum cut** (or just **min cut**) is a cut with the least total size.
  - Intuitively: removing the edges crossing a min cut is the easiest way to disconnect the graph.
Image Segmentation
Technically, this is the weighted min cut problem, but it's closely related to unweighted min cut.
Properties of Min Cuts
Properties of Min Cuts

- **Claim**: The size of a min cut is at most the minimum degree in the graph.
- If \( v \) has the minimum degree, then the cut \( (\{v\}, V - \{v\}) \) has size equal to \( \text{deg}(v) \).
- Since the minimum cut is no larger than any cut in the graph, this means that minimum cut has size at most \( \text{deg}(v) \) for any \( v \in V \).
Properties of Min Cuts

**Theorem:** In an $n$-node graph, if there is a min cut with cost $k$, there must be at least $nk / 2$ edges.

**Proof:** If there is a minimum cut with cost $k$, every node must have degree at least $k$ (since otherwise there would be a cut with cost less than $k$). Therefore, by the handshaking lemma, we have

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

And so $m \geq nk / 2$, as required. ■
Finding a Global Min Cut: 
Karger's Algorithm
Karger's Algorithm

- Given an edge \((u, v)\) in a multigraph, we can **contract** \(u\) and \(v\) as follows:
  - Delete all edges between \(u\) and \(v\).
  - Replace \(u\) and \(v\) with a new “supernode” \(uv\).
  - Replace all edges incident to \(u\) or \(v\) with edges incident to the supernode \(uv\).

- **Karger's algorithm** is as follows:
  - If there are exactly two nodes left, stop. The edges crossing those nodes form a cut.
  - Otherwise, pick a random edge, contract it, then repeat.
Karger's Algorithm

- Consider any cut $C = (S, V - S)$.
- If we never contract any edges crossing $C$, then Karger's algorithm will produce the cut $C$.
  - Initially, all nodes are in their own cluster.
  - Contracting an edge that does not cross the cut can only connect nodes that both belong to the same side of the cut.
  - Stops when two supernodes remain, which must be the sets $S$ and $V - S$. 
The Story So Far

- We now have the following:
  
  Karger's algorithm produces cut $C$ iff it never contracts an edge crossing $C$.

- How does this relate to min cuts?
- Across all cuts, min cuts have the lowest probability of having an edge contracted.
  - Fewer edges than all non-min cuts.
- Intuitively, we should be more likely to get a min cut than a non-min cut.
- What is the probability that we do get a min cut?
Defining Random Variables

• Choose any minimum cut $C$; let its size be $k$.
• Define the event $\mathcal{E}$ to be the event that Karger's algorithm produces $C$.
• This means that on each iteration, Karger's algorithm must not contract any of the edges crossing $C$.
• Let $\mathcal{E}_k$ be the event that on iteration $k$ of the algorithm, Karger's algorithm does not contract an edge crossing $C$.
• Then $\mathcal{E} = \bigcap_{i=1}^{n-2} \mathcal{E}_i$

Can anyone explain the summation bounds?
Evaluating the Probability

- We want to know

\[
P(\mathcal{E}) = P\left( \bigcap_{i=1}^{n-2} \mathcal{E}_i \right)
\]

- These events are *not* independent of one another. (*Why?*)

- By the **chain rule for conditional probability**:

\[
P\left( \bigcap_{i=1}^{n-2} \mathcal{E}_i \right) = P(\mathcal{E}_{n-2}|\mathcal{E}_{n-3}, \ldots, \mathcal{E}_1) P(\mathcal{E}_{n-3}|\mathcal{E}_{n-4}, \ldots, \mathcal{E}_1) \ldots P(\mathcal{E}_2|\mathcal{E}_1) P(\mathcal{E}_1)
\]
The First Iteration

- First, let's evaluate $P(\bar{E}_1)$, the probability that we don't contract an edge from $C$.
- For simplicity, we'll evaluate $P(\bar{E}_1)$, the probability we do contract an edge from $C$ on the first round.
- If our min cut has $k$ edges, the probability that we choose one of the edges from $C$ is given by $k/m$.
- Since the min cut has $k$ edges, $m \geq kn/2$. Therefore:

$$P(\bar{E}_1) = \frac{k}{m} \leq \frac{k}{nk/2} = \frac{2}{n}$$

- So

$$P(E_1) = 1 - P(\bar{E}_1) \geq 1 - \frac{2}{n} = \frac{n-2}{n}$$
Successive Iterations

- We now need to determine
  \[ P(\mathcal{E}_i | \mathcal{E}_{i-1} \mathcal{E}_{i-2} \ldots \mathcal{E}_1) \]
- This is the probability that we don't contract an edge in \( C \) in round \( i \), given that we haven't contracted any edge in \( C \) at this point.
- As before, we'll look at the complement of this event:
  \[ P(\bar{\mathcal{E}}_i | \mathcal{E}_{i-1} \mathcal{E}_{i-2} \ldots \mathcal{E}_1) \]
- This is the probability we do contract an edge from \( C \) in round \( i \) given that we haven't contracted any edges before this.
Successive Iterations

- At iteration $i$, $n - i + 1$ supernodes remain.
- **Claim:** Any cut in the contracted graph is also a cut in the original graph.
- Since $C$ has size $k$, all $n - i + 1$ supernodes must have at least $k$ incident edges. (*Why?)
- Total number of edges at least $k(n - i + 1) / 2$.
- Probability we contract an edge from $C$ is
  $$P(\bar{\mathcal{E}}_i | \mathcal{E}_{i-1} \mathcal{E}_{i-2} \ldots \mathcal{E}_1) \leq \frac{k}{k(n - i + 1)/2} = \frac{2}{n - i + 1}$$
- So
  $$P(\mathcal{E}_i | \mathcal{E}_{i-1} \mathcal{E}_{i-2} \ldots \mathcal{E}_1) \geq 1 - \frac{2}{n - i + 1} = \frac{n - i - 1}{n - i + 1}$$
\[ P(\mathcal{E}) = P(\mathcal{E}_{n-2} | \mathcal{E}_{n-3}, \ldots, \mathcal{E}_1) \ldots P(\mathcal{E}_2 | \mathcal{E}_1) P(\mathcal{E}_1) \]

\[ \geq \frac{n-(n-2)-1}{n-(n-2)+1} \cdot \frac{n-(n-3)-1}{n-(n-3)+1} \ldots \frac{n-2}{n} \]

\[ = \frac{1 \cdot 2 \cdots n-2}{3 \cdot 4 \cdots n} \]

\[ = \prod_{i=1}^{n-2} \frac{i}{i+2} \]

\[ = \frac{\prod_{i=1}^{n-2} i}{\prod_{i=1}^{n-2} i+2} \]

\[ = \frac{\prod_{i=1}^{n-2} i}{\prod_{i=3}^{n} i} \]

\[ = \left(1 \cdot 2 \cdot \prod_{i=3}^{n-2} i\right) / \left(n \cdot (n-1) \cdot \prod_{i=3}^{n-2} i\right) \]

\[ = \frac{2}{n(n-1)} \]
The Success Probability

- Right now, the probability that the algorithm finds a minimum cut is at least
  \[ \frac{2}{n(n-1)} \]
- This number is low, but it's not as low as it might seem.
  - How many total cuts are there?
  - If we picked a cut randomly and there was just one min cut, what's the probability that we would find it?
Amplifying the Probability

- Recall: running an algorithm multiple times and taking the best result can amplify the success probability.

- Run Karger's algorithm for $k$ iterations and take the smallest cut found. What is the probability that we don't get a minimum cut?

$\left(1 - \frac{2}{n(n-1)}\right)^k$
A Useful Inequality

- For any $x \geq 1$, we have
  \[ \frac{1}{4} \leq \left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e} \]

- If we run Karger's algorithm $n(n-1)/2$ times, the probability we don't get a minimum cut is
  \[ \left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)/2} \leq \frac{1}{e} \]

- If we run Karger's algorithm $(n(n-1)/2) \ln n$ times, the probability we don't get a minimum cut is
  \[ \left(1 - \frac{2}{n(n-1)}\right)^{(n(n-1)/2)\ln n} \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n} \]
The Overall Result

- Running Karger's algorithm $O(n^2 \log n)$ times produces a minimum cut with high probability.

- **Claim:** Using adjacency matrices, it's possible to run Karger's algorithm once in time $O(n^2)$.

- **Theorem:** Running Karger's algorithm $O(n^2 \log n)$ times gives a minimum cut with high probability and takes time $O(n^4 \log n)$. 
Speeding Things Up:
The Karger-Stein Algorithm
Some Quick History

• David Karger developed the contraction algorithm in 1993. Its runtime was $O(n^4 \log n)$.

• In 1996, David Karger and Clifford Stein (the “S” in CLRS) published an improved version of the algorithm that is dramatically faster.

• **The Good News:** The algorithm makes intuitive sense.

• **The Bad News:** Some of the math is really, really hard.
Some Observations

- Karger's algorithm only fails if it contracts an edge in the min cut.
- The probability of contracting the wrong edge increases as the number of supernodes decreases.
  - *(Why?)*
- Since failures are more likely later in the algorithm, repeat just the later stages of the algorithm when the algorithm fails.
Intelligent Restarts

- Interesting fact: If we contract from $n$ nodes down to $n/\sqrt{2}$ nodes, the probability that we don't contract an edge in the min cut is about 50%.
  - Can work out the math yourself if you'd like.
- What happens if we do the following?
  - Contract down to $n/\sqrt{2}$ nodes.
  - Run *two* passes of the contraction algorithm from this point.
  - Take the better of the two cuts.
The Success Probability

- This algorithm finds a min cut iff
  - The partial contraction step doesn't contract an edge in the min cut, and
  - At least one of the two remaining contractions does find a min cut.
- The first step succeeds with probability around 50%.
- Each remaining call succeeds with probability at least \(4 / n(n - 1)\).
  - (Why?)
The Success Probability

$$P(\text{success}) \geq \frac{1}{2}\left(1-\left(1-\frac{4}{n(n-1)}\right)^2\right)$$

$$= \frac{1}{2}\left(1-\left(1-\frac{8}{n(n-1)}+\frac{16}{n^2(n-1)^2}\right)\right)$$

$$= \frac{1}{2}\left(\frac{8}{n(n-1)}-\frac{16}{n^2(n-1)^2}\right)$$

$$= \frac{4}{n(n-1)}-\frac{8}{n^2(n-1)^2}$$
A Success Story

- This new algorithm has roughly twice the success probability as the original algorithm!

- **Key Insight:** Keep repeating this process!
  - Base case: When size is some small constant, just brute-force the answer.
  - Otherwise, contract down to $n/\sqrt{2}$ nodes, then recursively apply this algorithm twice to the remaining graph and take the better of the two results.

- This is the **Karger-Stein** algorithm.
Two Questions

• What is the success probability of this new algorithm?
  • This is extremely difficult to determine.
  • We'll talk about it later.

• What is the runtime of this new algorithm?
  • Let's use the Master Theorem?
The Runtime

- We have the following recurrence relation:

\[
T(n) = \begin{cases} 
  c & \text{if } n \leq n_0 \\
  2T\left(\frac{n}{\sqrt{2}}\right) + O(n^2) & \text{otherwise}
\end{cases}
\]

- What does the Master Theorem say about it?

\[T(n) = O(n^2 \log n)\]
The Accuracy

- By solving a very tricky recurrence relation, we can show that this algorithm returns a min cut with probability $\Omega(1 / \log n)$.

- If we run the algorithm roughly $\ln^2 n$ times, the probability that all runs fail is roughly

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

- **Theorem:** The Karger-Stein algorithm is an $O(n^2 \log^3 n)$-time algorithm for finding a min cut with high probability.
Major Ideas from Today

- You can increase the success rate of a Monte Carlo algorithm by iterating it multiple times and taking the best option found.
  - If the probability of success is $1 / f(n)$, then running it $O(f(n) \log n)$ times gives a high probability of success.
  - If you're more intelligent about how you iterate the algorithm, you can often do much better than this.
Next Time

- Hash Tables
- Universal Hashing