CS 161	Lecture 16: Min Cut and Karger's Algorithm
Winter 2021	Wed, Mar 10

Adapted from Virginia Williams's lecture notes. Additional credits go to Peng Hui How, Anthony Kim and Mary Wootters.

Please direct all typos and mistakes to Moses Charikar and Nima Anari.

---

1 Minimum Cut Problem

Today, we introduce the minimum cut problem. This problem has many motivations, one of which comes from image segmentation. Imagine that we have an image made up of pixels – we want to partition the image into two dissimilar portions. If we think of the pixels as nodes in the graph and add in edges between similar pixels, the min cut will correspond to a partition of the pixels where the two parts are most dissimilar.

Let us start with the definition of a cut. A *cut* S of a graph $G = (V, E)$ is a proper subset of V ($S \subset V$ and $S \neq \emptyset, V$). The size of a cut with respect to S is the number of edges between S and the rest of the graph $\bar{S} = V \setminus S$. In the example below, the size of the cut defined by the set S of black nodes and set $V \setminus S$ of white nodes is 2.

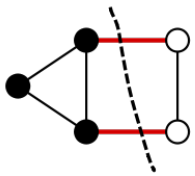


Figure 1: Cut size = 2 (Image Source: Wikipedia)

The minimum cut problem (abbreviated as "min cut"), is defined as follows:

Input: Undirected graph $G = (V, E)$

Output: A minimum cut S, that is, a partition of the nodes of G into S and $V \setminus S$ that minimizes the number of edges going across the partition. Intuitively, we want to "destroy" the smallest number of edges possible in the image segmentation problem.

Let n be the number of vertices and m be the number of edges. For finding the min-cut, a brute-force solution is to enumerate over all $O(2^n)$ subsets. There is also a flow-based algorithm using the well-known Max-Flow Min-Cut Theorem which we describe below. However, these algorithms are still inefficient. We present a more efficient algorithm, Karger's algorithm, in the next section.

1.1 Connections to Minimum s-t Cut and Maximum Flow

Next lecture we'll talk about a related problem, that of a Minimum s-t cut. We define the minimum s-t cut problem as follows:

Input: Undirected graph $G = (V, E)$, and vertices s and t

Output: A minimum cut S that separates s and t, that is, a partition of the nodes of G into S and $V \setminus S$ with $s \in S$ and $t \in V \setminus S$ that minimizes the number of edges going across the partition.

The minimum s-t cut is closely related to the maximum flow from s to t. We consider the pipe network given by G where the vertices form the junctures and the edges form the pipes of uniform capacity. The maximum flow from s to t is the maximum amount of flow that can be sent through the pipe network from source s to sink t subject to the capacity constraints. In fact, the *Max-Flow Min-Cut Theorem* states that the minimum s-t cut and the maximum flow amount are equal. Equivalently, finding a solution to one problem will lead to a solution to the other problem, and vice versa. Intuitively, when no more flow can be routed, we have found a minimum s-t cut.

We note that the maximum flow problem can be solved in polynomial time. Using the algorithm for finding the maximum flow as a black-box, we can solve the minimum cut problem as follows. We arbitrarily fix a node to be s. For each remaining vertex as t, we find the maximum flow from s to t and the corresponding minimum s-t cut. We choose the best of these minimum s-t cuts to be the minimum cut. The algorithm is correct since the optimal minimum cut is a minimum s-t cut for some nodes s and t and the algorithm iterates over all these possibilities. The run time would be polynomial in n and m and depend on the black-box routine for solving the maximum flow.

2 Karger's Algorithm

2.1 Las Vegas and Monte Carlo Algorithms

Karger's algorithm is a randomized algorithm. It is different from the randomized algorithms that we have seen before. The randomized algorithms we have seen so far (such as Quicksort) have good running time in expectation, but may occasionally run for significantly longer. Nevertheless, these algorithms always produce a correct solution upon termination (or report that they failed). Algorithms with the properties above are known as “Las Vegas" algorithms.

This is not the case for Karger's algorithm. Karger's algorithm is a randomized algorithm whose run time is deterministic; that is, on every run, we can bound the run time using a function in the size of the input that will not depend on the random choices made by the algorithm, but the algorithm may return a wrong answer with a small probability. Such an algorithm is called a “Monte Carlo" algorithm.

2.2 Finding a Min-Cut

Karger's algorithm will use the notions of "supernodes" and "superedges". A supernode is a group of nodes. A superedge connecting two supernodes X and Y consists of all edges between a pair of nodes, one from X and one from Y. Initially, all nodes will start as their own supernode and every superedge just contains a single edge. The intuition behind Karger's Algorithm is to pick any edge at random (among all edges), merge its endpoints, and repeat the process until there are only two supernodes left. These supernodes will define the cut.

Algorithm 1: IntuitiveKarger(G)

while *there are more than 2 supernodes:* **do**
    Pick an edge $(u, v) \in E(G)$ uniformly at random
    Merge u and v
Output edges between the remaining two supernodes

The goal of this lecture will be to show that this simple algorithm can be made to work with good probability. We use the following notations: We will refer to the set of nodes within a supernode u as $V(u)$, and the set of edges between two supernodes u, v as E_{uv} (this is the superedge between u and v). To distinguish the singleton nodes and supernodes, we use u to denote a singleton node and $\bar{u}$ to denote a supernode when necessary.

The initialization step of Karger's algorithm is as follows:

Algorithm 2: Initialize(G)

$\Gamma \leftarrow \emptyset$; // the set of supernodes
$F \leftarrow \emptyset$; // the set of superedges
foreach $v \in V$ **do**
    $\bar{v} \leftarrow$ new supernode
    $V(\bar{v}) \leftarrow \{v\}$
    $\Gamma \leftarrow \Gamma \cup \{\bar{v}\}$
foreach $(u, v) \in E$ **do**
    $E_{uv} \leftarrow \{(u, v)\}$
    $F \leftarrow F \cup \{(u, v)\}$

We merge two supernodes as follows:

Algorithm 3: Merge(a, b, Γ) //Γ is the set of supernodes with $a, b \in \Gamma$

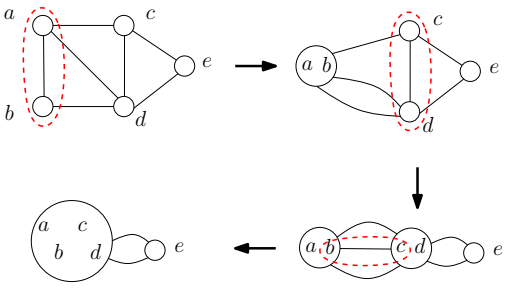
$x \leftarrow$ new supernode
$V(x) \leftarrow V(a) \cup V(b)$; //merge the vertices of a and b
foreach $d \in \Gamma \backslash \{a, b\}$ **do**
    // $O(n)$ iterations
    $E_{xd} \leftarrow E_{ad} \cup E_{bd}$; //$O(1)$ operation using linked lists
$\Gamma \leftarrow (\Gamma \backslash \{a, b\}) \cup \{x\}$;

Now, we can present the Karger's algorithm in full detail.

Algorithm 4: Karger(G)

Initialize(G); //Γ is the set of supernodes, F is the set of edges that are part of the current superedges
while $|\Gamma| > 2$ **do**
    $(u, v) \leftarrow$ uniform random edge from F
    Merge($\bar{u}, \bar{v}, \Gamma$); // Note $u \in \bar{u}, v \in \bar{v}$
    $F \leftarrow F \backslash E_{\bar{u}\bar{v}}$
Return one of the supernodes in Γ and $|E_{xy}|$; //$\Gamma = \{x, y\}$

The following example illustrates one possible execution of Karger's Algorithm.



The example happens to give the correct minimum cut, but only because we carefully picked the edges for merging. There are many other choices of edges for merging, so it's possible we could have ended with a cut with more than 2 edges.

The run time of the algorithm is $O(n^2)$ since each merge operation takes $O(n)$ time (going through at most $O(n)$ edges and vertices), and there are $n - 2$ merges until there are 2 supernodes left. We can get a better run time using the union-find data structure used for Kruskal's algorithm. In each iteration, we randomly select an edge and merge the two supernodes on the edge in amortized time $O(\alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function. Since there are m edges, there are at most m merging operations. For the random selection of edges, we generate a random permutation of edges in $O(m)$ in the beginning and process the edges in that order. In total, the run time is $O(m\alpha(n))$ with the union-find data structure.

3 Analysis

We prove that we obtain the correct answer with sufficiently high probability under uniformly random selection of edges.

Claim 1. *The probability that Karger's algorithm returns a min-cut is at least* $\frac{1}{\binom{n}{2}}$.

Proof. Fix a particular min-cut S^*. If Karger's algorithm picks any edge across this cut to do a merge on, then S^* will not be output. However, if all edges that the algorithm selects are

not across the cut, then S^* will be output. Let $e_1, \ldots, e_{n-2}$ be the random edges selected by Karger's algorithm for the merging operations, i.e., those edges that decreased the number of supernodes. Then,

$$\begin{aligned}
&\Pr(\text{Karger's algorithm outputs } S^*) \\
&= \Pr(e_1 \text{ does not cross } S^*) \cdot \Pr(e_2 \text{ does not cross } S^* \mid e_1 \text{ does not cross } S^*) \cdot \\
&\quad \cdots \Pr(e_{n-2} \text{ does not cross } S^* \mid e_1, \ldots, e_{n-3} \text{ do not cross } S^*) \,.
\end{aligned}$$

For the sake of analysis, we say that a cut is *alive* if it does not cut a supernode and that an edge is *alive* if its endpoints are in different supernodes. Note the definitions are with respect to the current multigraph maintained by Karger's algorithm. Equivalently, we can write the probability that Karger's algorithm outputs S^* as follows:

$$\begin{aligned}
&\Pr(\text{Karger's algorithm outputs } S^*) \\
&= \Pr(S^* \text{ is alive after merging on } e_1) \cdot \\
&\quad \Pr(S^* \text{ is alive after merging on } e_2 \mid S^* \text{ is alive after merging on } e_1) \cdot \\
&\quad \cdots \Pr(S^* \text{ is alive after merging on } e_{n-2} \mid S^* \text{ is alive after merging on } e_1, \ldots, e_{n-3}) \,.
\end{aligned}$$

Suppose the size of the minimum cut S^* is k. Then, the graph has at least $\frac{nk}{2}$ edges. This is because each node has a degree at least k and the number of edges is exactly a half of the total degrees of the nodes. Similarly, if the min-cut S^* is alive with respect to the current multigraph with, say, t supernodes, there are at least $\frac{tk}{2}$ edges that are alive. In particular, if S^* is alive then the size of any min-cut of the multigraph is still k. Each supernode has at least k alive edges leaving it and the lower bound on the total number of alive edges follows.

For $i = 1, \ldots, n-2$, we compute

$$p_i = \Pr(S^* \text{ is alive after merging on } e_i \mid S^* \text{ is alive after merging on } e_1, \ldots, e_{i-1}) \,.$$

Given that S^* is alive after merging on $e_1, \ldots, e_{i-1}$, we know that there are $n-i+1$ supernodes and at least $\frac{(n-i+1)k}{2}$ edges are alive after $i-1$ merging operations. S^* is alive after merging on e_i if e_i is not one of the k edges in the minimum cut S^*. Hence, the probability that S^* is alive merging on e_i is

$$p_i \geq 1 - \frac{k}{(n-i+1)k/2} = 1 - \frac{2}{n-i+1} = \frac{n-i-1}{n-i+1} \,.$$

Then, the probability that Karger's algorithm outputs S^* is at least

$$\begin{aligned}
\Pr(\text{Karger's algorithm outputs } S^*) &= p_1 \cdot p_2 \cdots p_{n-2} \\
&\geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3} \\
&= \frac{2}{n(n-1)} \,,
\end{aligned}$$

where the last equality follows from cancellations except the first two denominators and last two numerators.

As S^* was arbitrary and there exists at least one minimum cut by definition, Karger's algorithm returns a min-cut with probability at least $\frac{2}{n(n-1)}$. □

The success probability of $1/\Theta(n^2)$ might seem very small. However, we'll see that we can boost this probability to an arbitrarily large probability by performing many independent trials of Karger's algorithm. Let $C > 0$ be an arbitrarily large constant.

Algorithm 5: AmplifiedKarger(G) //C is a constant

Run $C \cdot \binom{n}{2} \cdot \ln n$ independent Karger procedures.
Return the best of the cuts computed so far.

The run time of AmplifiedKarger is clearly $O(n^4 \log n)$ since we run $O(n^2 \log n)$ trials of an $O(n^2)$ time algorithm.

Claim 2.

$$\Pr(\text{AmplifiedKarger } \textit{is correct}) \geq 1 - \frac{1}{n^C}$$

where C is the constant used in the amplification algorithm.

Remark 3 (Useful fact). For any real x, $1 + x \leq e^x$.

Proof of Claim 2.

$$\begin{aligned}
\Pr(\text{AmplifiedKarger is incorrect}) &= \Pr(\text{Karger is incorrect for all the } C \cdot \binom{n}{2} \cdot \ln n \text{ independent runs}) \\
&= (\Pr(\text{Karger is incorrect}))^{C \cdot \binom{n}{2} \cdot \ln n} \\
&\leq \left(1 - \frac{2}{n(n-1)}\right)^{C \cdot \binom{n}{2} \cdot \ln n} \\
&\leq \exp\left(-\frac{2}{n(n-1)} C \binom{n}{2} \ln n\right) \\
&= \exp\left(-C \ln n\right) \\
&= \frac{1}{n^C}.
\end{aligned}$$

□

More generally, we can repeat a Monte Carlo algorithm with success probability of p for $\frac{1}{p} \ln\left(\frac{1}{\delta}\right)$ rounds to obtain the overall failure probability at most δ. The probability that all the

rounds fail is $(1-p)^{\frac{1}{p}\ln\left(\frac{1}{\delta}\right)}$ which can be upper bounded as follows:

$$\begin{aligned}(1-p)^{\frac{1}{p}\ln\left(\frac{1}{\delta}\right)} &\leq \exp\left(-p\frac{1}{p}\ln\left(\frac{1}{\delta}\right)\right)\\ &= \delta\,.\end{aligned}$$

Remark 4 (General Way of Boosting the Success Rate of Monte Carlo Algorithms). If an algorithm is correct with probability p, we can run it $\frac{1}{p}\ln\left(\frac{1}{\delta}\right)$ times and output the best result found, so that the amplified algorithm is correct with probability at least $1-\delta$.

In the proof of Claim 1, we actually proved a stronger statement that for any min-cut of G, Karger's algorithm returns that particular min-cut with probability at least $\frac{2}{n(n-1)}$. Interestingly, this implies that there can be at most $\binom{n}{2}$ cuts that have the minimum size.

Corollary 5. *There can be at most* $\binom{n}{2}$ *min-cuts.*

Proof. Let $C_1, C_2, \ldots$ be the complete enumeration of all possible cuts of G. Let $p_1, p_2, \ldots$ be the probabilities such that p_i is the probability that Karger's algorithm returns cut C_i. Without loss of generality, we assume that there are l cuts with the minimum size and they are $C_1, \ldots, C_l$. From the proof of Claim 1, it follows that $p_i \geq \frac{2}{n(n-1)}$ for $i = 1, \ldots, l$. Then,

$$1 = \sum_i p_i \geq \sum_{i=1}^{l} p_i \geq l \cdot \frac{2}{n(n-1)}\,.$$

The number of min-cuts l is at most $\frac{n(n-1)}{2}$. □

4 Karger-Stein Algorithm

The overall run time of the repeated application of Karger's algorithm is $O(n^4 \log n)$, without using the union-find data structure. We can further improve the run time of Karger's algorithm. Note that earlier merging operations are less risky in the sense that a particular min-cut is more likely to be alive after a merge operation since there are more alive edges. In particular, the probability that a minimum cut is alive until when there are t supernodes is $\frac{t(t-1)}{n(n-1)}$. For $t = \frac{n}{\sqrt{2}}$, this probability is approximately $\frac{1}{2}$. Essentially, it is highly likely that a min-cut is alive even after the first $n-t$ merging operations (such that t supernodes remain). To reduce the amount of work, it makes sense to repeat the random merging operations from the intermediate multigraph with t supernodes, rather than from the beginning.

In fact, the Karger-Stein algorithm relies on this idea and leads to a better overall run time than running Karger's algorithm n^2 times independently. We present the Karger-Stein algorithm below:

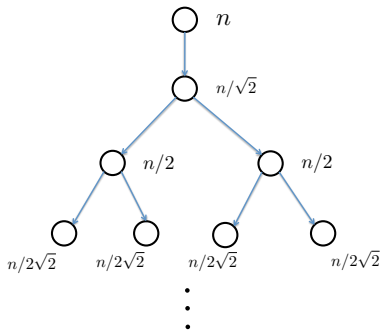
The following bound can be shown for the probability of success of Karger-Stein algorithm.

Claim 6. $\Pr(\textit{Karger-Stein is correct}) \geq \frac{1}{\Theta(\log n)}$.

Algorithm 6: KargerStein(G)

$n \leftarrow |G|$
if $n < 2\sqrt{2}$ **then**
    Find a min-cut by brute-force in $O(1)$
else
    Run Karger's algorithm on G until $\frac{n}{\sqrt{2}}$ supernodes remain;
    Let G_1 and G_2 be the two copies of the resulting multigraph
    $S_1 \leftarrow$ KargerStein(G_1)
    $S_2 \leftarrow$ KargerStein(G_2)
    Return the best of cuts S_1 and S_2

Proof. For a fixed min-cut S^*, we compute the probability that Karger-Stein algorithm returns S^* using a binary tree representation of the algorithm's computation. Let the nodes represent the multigraphs with some particular number of supernodes from intermediate stages of the algorithm. Let edges represent the series of random merging operations to reduce the number of supernodes in the "parent" multigraph by a factor of $\sqrt{2}$. The root is the original graph with n nodes, and it has one child that represents a multigraph of size $\frac{n}{\sqrt{2}}$. The child has two children each with $\frac{n}{2}$ supernodes. Each internal node below has two children. Note the tree has depth $2 \log n$ and has n^2 leaves. See the figure below.



We interpret the probability that Karger-Stein algorithm returns S^* as the probability that there exists a path from the root to a leaf using only "surviving" edges where each edge gets destroyed with probability $\frac{1}{2}$ independently. Let p_d be the probability that there exists a path consisting of surviving edges from a node at height d to a leaf. Note the height of a node is

measured from the bottom (the leaves). Then,

$$\begin{aligned}
p_d &= \frac{1}{2} \cdot \Pr(\exists \text{ a path of surviving edges in at least one subtree}) \\
&= \frac{1}{2} \cdot \big(\Pr(\exists \text{ a path of surviving edges in the left subtree}) + \\
&\qquad \Pr(\exists \text{ a path of surviving edges in the right subtree}) - \\
&\qquad \Pr(\exists \text{ a path of surviving edges in both left and right subtrees}) \big) \\
&= \frac{1}{2} \left(2p_{d-1} - (p_{d-1})^2\right) \\
&= p_{d-1} - \frac{1}{2}(p_{d-1})^2 \,.
\end{aligned}$$

We prove $p_d \geq \frac{1}{d+1}$ using induction. For the base case, $p_0 = 1$. For the inductive step, we note that if $p_{d-1} \geq \frac{1}{d}$, then

$$\begin{aligned}
p_d &= p_{d-1} - \frac{1}{2}(p_{d-1})^2 \\
&\geq \frac{1}{d} - \frac{1}{2}\frac{1}{d^2} \\
&\geq \frac{1}{d} - \frac{1}{d(d+1)} \\
&= \frac{1}{d+1} \,,
\end{aligned}$$

where the first inequality follows from the fact that $f(x) = x - \frac{1}{2}x^2$ is increasing for $x \in [0, 1]$ and the second inequality follows from $d \geq 1$.

As the root is at height $2 \log n$, the probability that Karger-Stein algorithm returns S^* is $p_{2 \log n} \geq \frac{1}{2 \log n + 1}$. □

We now argue about the run time.

Claim 7. *Karger-Stein algorithm runs in $O(n^2 \log n)$ time.*

Proof. On a multigraph with n supernodes, Karger-Stein algorithm does $O(n^2)$ amount of work to reduce the number of supernodes by a factor of $\sqrt{2}$ and also $O(n^2)$ work to determine the best of two cuts returned by recursive calls. It makes 2 recursive calls on multigraphs with $\frac{n}{\sqrt{2}}$ supernodes each. The run time $T(n)$ satisfies the following recurrence:

$$T(n) = 2T(n/\sqrt{2}) + O(n^2) \,.$$

By Master's Theorem, $T(n) = \Theta(n^2 \log n)$. □

Using the amplifying strategy, we can repeat Karger-Stein algorithm $O(\log n)$ times to obtain the overall failure rate at most some small constant. The overall run time of the amplified version of Karger-Stein algorithm is $O(n^2 \log^2 n)$ which is better than the amplified version of Karger's algorithm.