princeton univ. F'23 cos 521: Advanced Algorithm Design

Lecture 1: Karger's Min Cut Algorithm

Lecturer: *Huacheng Yu* Scribe:

Today's topic is simple but gorgeous: Karger's min cut algorithm and its extension. It is a simple randomized algorithm for finding the *global minimum cut* in an undirected graph: a (non-empty) subset of vertices S in which the set of edges leaving S, denoted $E(S, \overline{S})$ has minimum size among all subsets. You may have seen an algorithm for this problem in your undergrad class that uses maximum flow. Karger's algorithm is elementary and and a great introduction to randomized algorithms. For simplicity, we will consider unweighted graphs (the algorithm also works for weighted graphs). A more general problem is called min-k-cut, which asks to find the partition of the vertices into k disjoint sets that has the minimum total number of edges across the parts. Global minimum cut is simply the special case with $k = 2$. The algorithm we talk today also generalizes to min-k-cut naturally, although we will focus on the minimum cut problem.

1 Basic Operations

Karger's algorithm makes use of the following basic operations.

1. **Select a random edge:** For this lecture (and most lectures), we won't stress about how the algorithm accomplishes this. For instance, the algorithm might have the ability to select a random number between 1 and $m = |E|$, and can select the edge indexed by the random number. In any case, we will assume the algorithm can select a random edge in time $O(1)$.

2. **Contract an edge:** This operation takes two existing nodes in the graph with an edge between them and "merges" them into a super-node. That is, the operation Contract$(G = (V, E), e = \{u, v\})$ takes as input a graph G with vertex set V and edges E, and outputs a new graph G' with vertex set $V \setminus \{u, v\} \cup S_{u,v}$. That is, it replaces the two nodes u and v with a supernode $S_{u,v}$. The new edge set contains all edges between two nodes in $V \setminus \{u, v\}$. For all edges between u and $x \notin \{u, v\}$, add an edge between $S_{u,v}$ and x. Ditto for all edges between v and $x \notin \{u, v\}$. **Note that this may create multiple edges between two nodes and this is intended.** But do not include edges from $S_{u,v}$ to itself. This lecture, we will also not stress about how exactly to implement this operation, but note that it can be done in time $O(n)$.

2 The (Core) Algorithm

The algorithm is extremely simple: Pick a random edge, and contract it. Repeat until the graph has only two supernodes, which is output as our guess for min-cut. That is, if upon termination the two remaining nodes are S_X and $S_{\bar{X}}$, output $(X, \bar{X})$ as the guess for the minimum cut.

To guarantee a high success probability, re-run the algorithm from scratch k times independently, and output whichever guess $(X_1, \bar{X}_1), \ldots, (X_k, \bar{X}_k)$ is the smallest cut. k will be chosen shortly.

2.1 Intuition

Before we get into a formal proof (which itself is quite simple), here is some brief intuition. We say that a given cut $(X, \bar{X})$ *survives* contraction $e = \{u, v\}$ if $|X \cap \{u, v\}| \neq 1$. That is, a cut survies the contraction of edge e as long as edge e is between two nodes on the same side of the cut. The idea is that once an edge that crosses the cut $(X, \bar{X})$ is contracted, we have guaranteed that we cannot possibly output cut $(X, \bar{X})$ at the end. On the other hand, if we never contract an edge that crosses cut $(X, \bar{X})$, then $(X, \bar{X})$ will be exactly the cut we output.

So the idea is that we output a cut if and only if it survives all $n - 2$ contractions. At every step, the cut that is most likely to survive is exactly the global min-cut, exactly because it has fewer edges that "kill" it than all other cuts.

This algorithm also looks like a great heuristic to try on all kinds of real-life graphs, where one wants to *cluster* the nodes into "tightly-knit"portions. For example, social networks may cluster into communities; graphs capturing similarity of pixels may cluster to give different portions of the image (sky, grass, road etc.). Thus instead of continuing Karger's algorithm until you have two supernodes left, you could stop it when there are k supernodes and try to understand whether these correspond to a reasonable clustering.

3 Analysis

We begin with the following observation, which is really more of a definition than observation:

Observation 1
Let G' be obtained by a sequence of edge contractions of G. Then there is a one to one correspondence between cuts $(Y, \bar{Y})$ of G' and cuts $(X, \bar{X})$ of G that survived all contractions. Namely, the cut $(Y, \bar{Y})$ in G' corresponds to the cut $X = \cup_{y \in Y} S(y)$, where $S(y)$ denotes the original verticies of G that were contracted to form the super-node y in G'.

The key corollary in the analysis of Karger's algorithm follows the following simple lemma:

Lemma 1
Let G be an undirected graph, potentially with multi-edges but not self-loops, and let c be the value of the min-cut of G. Then $|E(G)| \geq nc/2$.

Proof: The cut $(\{v\}, E \setminus \{v\})$ is a potential min-cut, and has value exactly $d(v)$ (the degree of v). Therefore, $d(v) \geq c$ for all v. We can write $|E| = \sum_v d(v)/2 \geq nc/2$. □

Corollary 2
Let G be an undirected graph, potentially with multi-edges but not self-loops. Let $(X, \bar{X})$ be any minimum cut of G. Then the probability that $(X, \bar{X})$ survives the contraction of a random edge is at least $(1 - 2/n)$.

PROOF: By Lemma 1, there are at least $cn/2$ edges that might be selected. Exactly c of them would kill $(X, \bar{X})$. So the probability that $(X, \bar{X})$ survives is at least $1 - \frac{c}{cn/2} = 1 - 2/n$. □

And now we can conclude with the main theorem:

THEOREM 3
[Karger 1993] For any graph G, and any min-cut $(X, \bar{X})$ of G, Karger's algorithm outputs $(X, \bar{X})$ with probability at least $\frac{2}{n(n-1)}$.

PROOF: We know that Karger's algorithm outputs $(X, \bar{X})$ if and only if X survives every contraction. The probability that X survives the first contraction is $1 - 2/n$ by Corollary 2, and the probability that it survives the i^{th} contraction, conditioned on surviving the first $i - 1$ is $1 - 2/(n - i - 1)$ (also by Corollary 2). So the probability that it survives every contraction is at least:

$$\prod_{i=1}^{n-2}(1 - 2/(n - (i-1))) = \prod_{i=1}^{n-2}(n - i - 1)/(n - i + 1) = \frac{2}{n(n-1)}.$$

The last equality is due to a telescoping product. □

Note that in order to guarantee that the min-cut survived at least one iteration except with probability ϵ, we would need to repeat the procedure independently $\Theta(n^2 \ln(1/\epsilon))$ times. The runtime of each iteration is $O(n^2)$ because we do $n - 2$ contractions, each of which takes time $O(n)$, so the total runtime of the algorithm is $O(n^4)$, which is sad.

Still, notice that we have proved some cool facts: we showed that any particular min-cut is output with probability $\frac{2}{n(n-1)}$. As the sum of probabilities of disjoint events cannot exceed one, we have therefore shown that the number of global mincuts in a graph cannot exceed $\binom{n}{2}$. Note that there are initially 2^{n-1} possible cuts, so this is significantly smaller.

4 Improved Karger-Stein Algorithm

Karger and Stein improve the algorithm to run in time $O(n^2 \log^2(n))$ (essentially replacing two factors of n with $\log(n)$ instead). The idea is that roughly that *repetition ensures fault tolerance*. The real-life advice of making two backups of your hard drive is related to this: the probability that both fail is much smaller than one does. In case of Karger's algorithm, the overall probability of success is too low.

The main idea is this: we're actually unlikely to kill the min-cut in the first several contractions, so why are we repeating these every single time. Instead, we should be a little more clever about exactly which contractions we repeat.

OBSERVATION 2
Let G be an undirected graph with n nodes, possibly with multi-edges but not self-loops. Let $(X, \bar{X})$ be any minimum cut of G, then the probability that $(X, \bar{X})$ survives $n - n/\sqrt{2}$ random contractions is at least $1/2$.

PROOF: Exactly the same math as the proof of Theorem 3, except stop at $n - n/\sqrt{2}$ instead of $n - 2$. This telescopes to at least $1/2$. □

Now, consider the following recursive algorithm: starting from a graph G (with n nodes, multi-edges but no self-loops), randomly contract edges until only $n/\sqrt{2}$ nodes remain, and call the resulting graphs G'. Then, call the algorithm twice independently on G', and output the smaller of the two returned cuts.

The runtime of the algorithm satisfies the recurrence:

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

$T(n) = O(n^2 \log n)$ solves the recurrence.[1] So each independent run of Karger-Stein has total runtime barely more than Karger's algorithm itself, but the redundancy should guarantee a higher success rate. We again need to analyze the probability that the min-cut survives the Karger-Stein algorithm.

THEOREM 4 (KARGER-STEIN 1996)
The probability that Karger-Stein outputs a min-cut is $\Omega(1/\log n)$.

PROOF: Let $(X, \bar{X})$ be a min-cut of G with n vertices, define $P(n)$ to be the probability that *some leaf* in the recursion of Karger-Stein produces $(X, \bar{X})$. We see that Karger-Stein produces $(X, \bar{X})$ if and only if:

- A min-cut survives the first $n - n/\sqrt{2}$ contractions (this occurs with probability at least $1/2$, by Observation).
- At least one of the recursive calls succeeds. Each recursive call succeeds with probability at least $P(n/\sqrt{2})$, by definition.

So the probability of success for each try is at least $(1 - (1 - P(n/\sqrt{2}))^2)/2$, and we get $P(n) \geq (1 - (1 - P(n/\sqrt{2}))^2)/2$. The last step is again solving the recurrence, which is $\Omega(1/\log n)$.

To see this, assume for inductive hypothesis that $P(n/\sqrt{2}) \geq c/\log(n/\sqrt{2})$. Then we get:

$$\begin{aligned}
P(n) &\geq (1 - (1 - P(n/\sqrt{2}))^2)/2 \\
&\geq (1 - (1 - c/\log(n/\sqrt{2}))^2)/2 \\
&= \frac{c}{\log(n/\sqrt{2})} - \frac{c^2}{2\log^2(n/\sqrt{2}))} \\
&= \frac{c}{\log n - 1/2} - \frac{c^2}{2(\log n - 1/2)^2} \\
&= \frac{c}{\log n} + \frac{2c(\log n - 1/2)\log n - c^2 \log n - 2c(\log n - 1/2)^2}{2(\log n - 1/2)^2 \log n} \\
&= \frac{c}{\log n} + \frac{c\log n - c^2 \log n - c/2}{2(\log n - 1/2)^2 \log n} \\
&\geq \frac{c}{\log n}.
\end{aligned}$$

[1] Note that this can be done using the Master Theorem, or instead by writing a binary tree, observing that there are $O(\log n)$ levels, and that each level has total "excess" work of $O(n^2)$. On level i, there are 2^i nodes, each with "excess work" $O((n/\sqrt{2}^i)^2) = O(n^2/2^i)$.

The last inequality is true as long as $c < 1$ and n sufficiently large. So we get that the probability of success is $\Omega(1/\log(n))$. $\square$

Now, we can repeat the entire algorithm independently $O(\log(n)\log(1/\epsilon))$ times to get a total success rate of $1 - \epsilon$.

Note that if we run the entire algorithm $c\log^2 n$ times for a sufficiently large constant c, the analysis shows that for any min-cut, it is found in some leaf in some execution with probability

$$1 - (1 - 1/\log n)^{c\log^2 n} \geq 1 - 1/n^c.$$

Hence, by the fact that there are at most $O(n^2)$ min-cuts and by union bound, we will generate *every* min-cut in some leaf in some execution with high probability – this shows that the $O(n^2)$ min-cuts have some structure that allows us to implicitly encode succinctly.