CSE 190, Great ideas in algorithms: Karger's min-cut algorithm

1 Min cut

Let $G = (V, E)$ be a graph. A cut in this graph is $|E(S, S^c)|$ for some $S \subset V$, that is, the number of edges which cross a partition of the vertices. Finding the maximum cut is NP-hard, and the best algorithms solving it run in exponential time. We saw an algorithm which finds a 2-approximation for the max-cut. However, finding the minimum cut turns out to be solvable in polynomial time. Today, we will see a randomized algorithm due to Karger which achieves this is a beautiful way. To formalize this, the algorithm will find

$$\text{min-cut}(G) = \min_{\emptyset \neq S \subsetneq V} |E(S, S^c)|.$$

2 Karger's algorithm

The algorithm is very simple: choose a random edge and contract it. Repeat $n-1$ times, until only two vertices remain. Output this as a guess the for min-cut. We will show that this outputs the min-cut with probability at least $2/n^2$, hence repeating it $3n^2$ times (say) will yield a min-cut with probability at least $(1 - 2/n^2)^{3n^2} \geq 99\%$.

Formally, to define a contraction we need to allow graphs to have parallel edges.

Definition 2.1 (edge contraction). *Let $G = (V, E)$ be an undirected graph with $|V| = n$ vertices, potentially with parallel edges. For an edge $e = (u, v) \in E$, the contraction of G along e is an undirected graph on $n-1$ vertices, where we merge u, v to be a single node, and delete any self-loops that may be created in this process.*

We can now present the algorithm.

```
Karger

Input: Undirected graph $G = (V, E)$ with $|V| = n$
Output: Cut in $G$

1. Let $G_n = G$.
2. For $i = n, \ldots, 3$ do:
2.1 Choose a uniform edge $e_i \in G_i$.
2.2 Set $G_{i-1}$ to be the contraction of $G_i$ along $e_i$.
3. Output the cut in $G$ corresponding to $G_2$.
```

We make a few observations: by contracting along an edge $e = (u, v)$, we make a commitment that u, v belong to the same side of the cut, so we restrict the number of potential cuts. Hence, any cut of G_i is a cut of G, and in particular

$$\text{min-cut}(G) \leq \text{min-cut}(G_i), \quad i = n, \ldots, 2.$$

Fix a min-cut $S \subset V(G)$. We will analyze the algorithm by analyzing the probability that the algorithm never chooses an edge in S. Hence, after $n-1$ contractions, the output will be exactly the cut S. We do so by proving some claims.

Claim 2.2. *min-cut*$(G) \leq 2|E|/|V|$.

Proof. For any vertex $v \in V$, we can form a cut by taking $E(\{v\}, V \setminus \{v\})$. It has $\deg(v)$ many edges. Since $2|E| = \sum_{v \in V} \deg(v)$ we can bound

$$\text{min-cut}(G) \leq \min_{v \in V} \deg(v) \leq \frac{2|E|}{|V|}.$$

□

Claim 2.3. *Let $G = (V, E)$ be a graph, $S \subset V$ be a minimal cut in G. Let $e \in E$ be a uniform chosen edge. Then*

$$\Pr_e[e \in E(S, S^c)] \leq 1 - \frac{2}{|V|}.$$

Proof. We saw that $|E(S, S^c)| \leq 2|E|/|V|$. So, the probability that $e \in E(S, S^c)$ is at most $(2|E|/|V|)/|E| = 2/|V|$. □

Theorem 2.4. *For any min-cut S in G,* $\Pr[$*algorithm outputs* $S] \geq \frac{1}{\binom{n}{2}} \geq \frac{2}{n^2}$.

Proof. Fix a min-cut $S \subset V$. The algorithm outputs S if it never chooses an edge in S. So

$$\begin{aligned}\Pr[\text{algorithm outputs } S] &= \Pr[e_n, \ldots, e_3 \notin E(S, S^c)] \\ &= \prod_{i=n}^{3} \Pr[e_i \notin E(S, S^c) | e_n, \ldots, e_{i+1} \notin E(S, S^c)].\end{aligned}$$

To analyze this, assume that $e_n, \dots, e_{i+1} \notin E(S, S^c)$, so that S is still a cut in G_i. Thus, it is also a min-cut in G_i, and we proved that in this case, $\Pr[e_i \in E(S, S^c)] \leq 1/2|V(G_i)| = 1/2i$. So

$$\begin{aligned}\Pr[\text{algorithm outputs } S] &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \ldots \cdot \frac{2}{4} \cdot \frac{1}{3} \\ &= \frac{2}{n(n-1)}.\end{aligned}$$

□

We get a nice corollary: the number of min-cuts in G is at most $\binom{n}{2}$. Can you prove it directly?

3 Improving the running time

The time it takes to find a min-cut by Karger's algorithm described above is $\Omega(n^4)$: we need $\Omega(n^2)$ iterations to guarantee success with high probability. Every iteration requires $n - 2$ rounds of contraction, and every contraction takes $\Omega(n)$ time. We will see how to improve this running time to $O(n^2)$ due to Karger and Stein. The main observation guiding this is that most of the error comes when the graph becomes rather small; however, when the graphs are small, the running time is smaller. So, we can allow to run multiple instances for smaller graphs, which would boost the success probability without increasing the running time too much.

Fast-Karger

```
Input: Undirected graph $G = (V, E)$ with $|V| = n$
Output: Cut in $G$

0. If $n = 2$ output the only possible cut.
1. Let $G_n = G$ and $m = n/\sqrt{2}$.
2. For $i = n, \dots, m+1$ do:
2.1 Choose a uniform edge $e_i \in G_i$.
2.2 Set $G_{i-1}$ to be the contraction of $G_i$ along $e_i$.
3. Run recursively $S_i = \text{Fast-Karger}(G_i)$ for $i = 1, 2$.
4. Output $S \in \{S_1, S_2\}$ which minimizes $E(S, S^c)$.
```

Claim 3.1. *Fix a cut S. Run the Karger algorithm, but output G_m for $m = n/\sqrt{2}$. Then the probability that S is still a cut in G_m is at least $1/2$.*

Proof. Repeating the analysis we did, but stopping once we reach G_m, gives

$$\Pr[S \text{ is a cut of } G_m] = \Pr[e_n, \ldots, e_{m+1} \notin E(S, S^c)] \geq \frac{n-2}{n} \cdot \ldots \cdot \frac{m-1}{m+1} = \frac{m(m-1)}{n(n-1)}.$$

So, if we set $m = n/\sqrt{2}$, this probability is at least 1/2. □

Theorem 3.2. *Fast-Karger runs in time* $O(n^2 \log n)$ *and returns a minimal cut with probability at least* 1/4*.*

Proof. We first analyze the success probability. Let $P(n)$ be the probability that the algorithm succeeds on graphs of size n. We will prove by induction that $P(n) \geq 1/4$. We proved that

$$\Pr[\text{min-cut}(G_m) = \text{min-cut}(G)] \geq \frac{1}{2}.$$

So we have

$$P(n) \geq \frac{1}{2} \cdot (1 - P(m))^2 \geq \frac{1}{2} \cdot \left(\frac{3}{4}\right)^2 = \frac{9}{32} \geq \frac{1}{4}.$$

We next analyze the running time. Let $T(n)$ be the running time of the algorithm on graphs of size n. Then

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

This solves to $T(n) = O(n^2 \log n)$. □