princeton univ. F'13 cos 521: Advanced Algorithm Design

Lecture 2: Karger's Min Cut Algorithm

Lecturer: *Sanjeev Arora* Scribe:*Sanjeev*

Today's topic is simple but gorgeous: Karger's min cut algorithm and its extension. It is a simple randomized algorithm for finding the *minimum cut* in a graph: a 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.

The algorithm is this: Pick a random edge, and merge its endpoints into a single "supernode."Repeat until the graph has only two supernodes, which is output as our guess for min-cut. (As you continue, the supernodes may develop parallel edges; these are allowed. Selfloops are ignored.)

Note that if you pick a random edge, it is more likely to come from parts of the graph that contain more edges in the first place. Thus this algorithm 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.

Today we will first see that the above version of the algorithm yields the optimum min cut with probability at least $2/n^2$. Thus we can repeat it say $20n^2$ times, and output the smallest cut seen in any iteration. The probability that the optimum cut is not seen in any repetition is at most $(1 - 2/n^2)^{20n^2} < 0.01$.

Unfortunately, this simple version has running time about n^4 which is not great.

So then we see a better version with a simple tweak that brings the running time down to closer to n^2. 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. But if run part of the way until the graph has $n/\sqrt{2}$ supernodes, the chance that the mincut hasn't changed is at least $1/2$. So you make two independent runs that go down to $n/\sqrt{2}$ supernodes, and recursively solve both of these. Thus the expected number of instances that will yield the correct mincut is $2 \times \frac{1}{2} = 1$. (Unwrapping the recursion, you see that each instance of size $n/\sqrt{2}$ will generate two instances of size $n/2$, and so on.) Simple induction shows that this 2-wise repetition is enough to bring the probability of success above $1/\log n$.

As you might suspect, this is not the end of the story but improvements beyond this get more hairy. If anybody is interested I can give more pointers.

Also this algorithm forms the basis of other algorithms for other tasks. Again, talk to me for pointers.

CSE 103: Probability and statistics **Winter 2010**

Topic 4 — Randomized algorithms, II

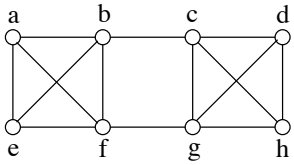
4.1 Karger's minimum cut algorithm

4.1.1 Clustering via graph cuts

Suppose a mail order company has the resources to prepare two different versions of its catalog, and it wishes to target each version towards a particular sector of its customer base. The data it has is a list of its regular customers, along with their purchase histories. How should this set of customers be partitioned into two coherent groups?

One way to do this is to create a graph with a node for each of the regular customers, and an edge between any two customers whose purchase patterns are similar. The goal is then to divide the nodes into two pieces which have very few edges between them.

More formally, the *minimum cut* of an undirected graph $G = (V, E)$ is a partition of the nodes into two groups V_1 and V_2 (that is, $V = V_1 \cup V_2$ and, $V_1 \cap V_2 = \emptyset$), so that the number of edges between V_1 and V_2 is minimized. In the graph below, for instance, the minimum cut has size two and partitions the nodes into $V_1 = \{a, b, e, f\}$ and $V_2 = \{c, d, g, h\}$.



4.1.2 Karger's algorithm

Here's a randomized algorithm for finding the minimum cut:

- Repeat until just two nodes remain:
  - Pick an edge of G at random and collapse its two endpoints into a single node
- For the two remaining nodes u_1 and u_2, set $V_1 = \{$nodes that went into $u_1\}$ and $V_2 = \{$nodes in $u_2\}$

An example is shown in Figure 4.1. Notice how some nodes end up having multiple edges between them.

4.1.3 Analysis

Karger's algorithm returns the minimum cut with a certain probability. To analyze it, let's go through a succession of key facts.

Fact 1. *If degree(u) denotes the number of edges touching node u, then*

$$\sum_{u \in V} \text{degree}(u) = 2|E|.$$

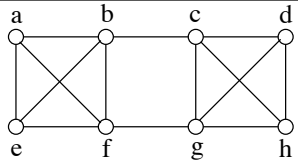
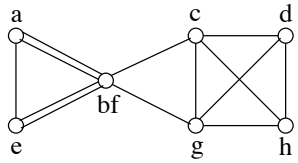
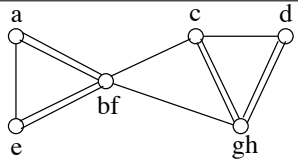
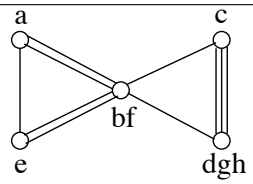
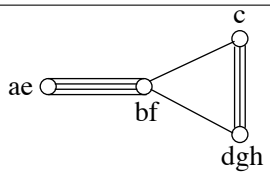
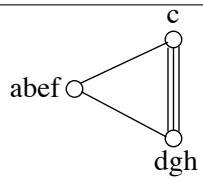
	14 edges to choose from Pick $b - f$ (probability 1/14)
	13 edges to choose from Pick $g - h$ (probability 1/13)
	12 edges to choose from Pick $d - gh$ (probability 1/6)
	10 edges to choose from Pick $a - e$ (probability 1/10)
	9 edges to choose from Pick $ab - ef$ (probability 4/9)
	5 edges to choose from Pick $c - dgh$ (probability 3/5)
	Done: just two nodes remain

Figure 4.1. Karger's algorithm at work.

To see this, imagine the following experiment: for each node, list all the edges touching it. The number of edges in this list is exactly the left-hand sum. But each edge appears exactly twice in it, once for each endpoint.

Fact 2. *If there are n nodes, then the average degree of a node is $2|E|/n$.*

This is a straightforward calculation: when you pick a node X at random,

$$\mathbb{E}[\text{degree}(X)] \;=\; \sum_{u \in V} \Pr(X = u)\text{degree}(u) \;=\; \frac{1}{n}\sum_{u} \text{degree}(u) \;=\; \frac{2|E|}{n}$$

where the last step uses the first Fact.

Fact 3. *The size of the minimum cut is at most $2|E|/n$.*

Consider the partition of V into two pieces, one containing a single node u, and the other containing the remaining $n-1$ nodes. The size of this cut is degree(u). Since this is a valid cut, the minimum cut cannot be bigger than this. In other words, for all nodes u,

$$(\text{size of minimum cut}) \leq \text{degree}(u).$$

This means that the size of the minimum cut is also $\leq$ the average degree, which we've seen is $2|E|/n$.

Fact 4. *If an edge is picked at random, the probability that it lies across the minimum cut is at most $2/n$.*

This is because there are $|E|$ edges to choose from, and at most $2|E|/n$ of them are in the minimum cut.

Now we have all the information we need to analyze Karger's algorithm. It returns the right answer *as long as it never picks an edge across the minimum cut.* If it always picks a non-cut edge, then this edge will connect two nodes on the same side of the cut, and so it is okay to collapse them together.

Each time an edge is collapsed, the number of nodes decreases by 1. Therefore,

$$\begin{aligned}
\Pr(\text{final cut is the minimum cut}) &= \Pr(\text{first selected edge is not in mincut}) \times \\
&\quad \Pr(\text{second selected edge is not in mincut}) \times \cdots \\
&\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right) \\
&= \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdots\frac{2}{4}\cdot\frac{1}{3} \\
&= \frac{2}{n(n-1)}.
\end{aligned}$$

The last equation comes from noticing that almost every numerator cancels with the denominator two fractions down the line.

Karger's algorithm succeeds with probabililty $p \geq 2/n^2$. Therefore, it should be run $\Omega(n^2)$ times, after which the smallest cut found should be chosen.

Those who are familiar with minimum spanning tree algorithms might be curious to hear that another way to implement Karger's algorithm is the following:

- Assign each edge a random weight
- Run Kruskal's algorithm to get the minimum spanning tree
- Break the largest edge in the tree to get the two clusters

(Do you see why?) Over the decades, the running time of Kruskal's algorithm has been thoroughly optimized via special data structures. Now this same technology can be put to work for cuts!

$$
\begin{aligned}
&= \frac{n-2}{n} \times \frac{n-3}{n-1} \times \cdots \times \frac{2}{4} \times \frac{1}{3} \\
&= \frac{2}{(n)(n-1)} \\
&= \frac{1}{\binom{n}{2}}.
\end{aligned}
$$

In order to boost the probability of success, we simply run the algorithm $\ell\binom{n}{2}$ times. The probability that at least one run succeeds is at least

$$1 - \left(1 - \frac{1}{\binom{n}{2}}\right)^{\ell\binom{n}{2}} \geq 1 - e^{-\ell}.$$

Setting $\ell = c \ln n$ we have error probability $\leq 1/n^c$. ■

It's easy to implement Karger's algorithm so that one run takes $O(n^2)$ time. Therefore, we have an $O(n^4 \log n)$ time randomized algorithm with error probability $1/\text{poly}(n)$.

A faster version of this algorithm was devised by Karger and Stein [4]. The key idea comes from looking at the telescoping product. In the initial contractions it's very unlikely we contracted an edge in the minimum cut. Towards the end of the algorithm, our probability of contracting an edge in the minimum cut grows.

From the earlier analysis we have the following. For a fixed minimum cut $\delta(S)$, the probability that this cut survives down to ℓ vertices is at least $\binom{\ell}{2}/\binom{n}{2}$. Thus, for $\ell = n/\sqrt{2}$ we have probability $\geq 1/2$ of succeeding. Hence, in expectation two trails should suffice.

Improved algorithm: From a multigraph G, if G has at least 6 vertices, repeat twice:

1. run the original algorithm down to $n/\sqrt{2} + 1$ vertices.
2. recurse on the resulting graph.

Return the minimum of the cuts found in the two recursive calls.

The choice of 6 as opposed to some other constant will only affect the running time by a constant factor.

We can easily compute the running time via the following recurrence (which is straightforward to solve, e.g., the standard Master theorem applies):

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

Since we succeed down to $n/\sqrt{2}$ with probability $\geq 1/2$, we have the following recurrence for the probability of success, denote by $P(n)$:

$$P(n) \geq 1 - \left(1 - \frac{1}{2}P(n/\sqrt{2}+1)\right)^2.$$

Inner term = Prob. that the subinstance preserved min cut AND recursive call found this optimum

Use

This solves to $P(n) = \Omega\left(\frac{1}{\log n}\right)$. Hence, similar to the earlier argument for the original algorithm, with $O(\log^2 n)$ runs of the algorithm, the probability of success is $\geq 1 - 1/poly(n)$.

1/(log n -0.5)

Therefore, in $O(n^2 \log^3 n)$ total time, we can find the minimum cut with probability $\geq 1 - 1/poly(n)$.

is approx.

Before finishing, we observe an interesting corollary of Karger's original algorithm which we will use in the next lecture to estimate the (un)reliability of a network.

1/log n + 0.5/(log n)^2

Corollary 5 *Any graph has at most $O(n^2)$ minimum cuts.*

This follows from Lemma 3 since that holds for any specified minimum cut.

Note, we can also enumerate all of these cuts by the above algorithm.

References

[1] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. Assoc. Comput. Mach.*, 35(4):921–940, 1988.

[2] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (Orlando, FL, 1992)*, pages 165–174, New York, 1992. ACM.

[3] D. R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (Austin, TX, 1993)*, pages 21–30, New York, 1993. ACM.

[4] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996.