

Jaques: *But, for the seventh cause; how did you find the quarrel on the seventh cause?*

Touchstone: *Upon a lie seven times removed:—bear your body more seeming, Audrey:—as thus, sir. I did dislike the cut of a certain courtier's beard: he sent me word, if I said his beard was not cut well, he was in the mind it was: this is called the Retort Courteous. If I sent him word again 'it was not well cut,' he would send me word, he cut it to please himself: this is called the Quip Modest. If again 'it was not well cut,' he disabled my judgment: this is called the Reply Churlish. If again 'it was not well cut,' he would answer, I spake not true: this is called the Reproof Valiant. If again 'it was not well cut,' he would say I lied: this is called the Counter-cheque Quarrelsome: and so to the Lie Circumstantial and the Lie Direct.*

Jaques: *And how oft did you say his beard was not well cut?*

Touchstone: *I durst go no further than the Lie Circumstantial, nor he durst not give me the Lie Direct; and so we measured swords and parted.*

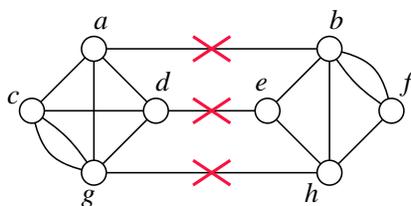
— William Shakespeare, *As You Like It* Act V, Scene 4 (1600)

5 Randomized Minimum Cut (September 19)

5.1 Setting Up the Problem

This lecture considers a problem that arises in robust network design. Suppose we have a connected multigraph¹ G representing a communications network like the UIUC telephone system, the internet, or Al-Qaeda. In order to disrupt the network, an enemy agent plans to remove some of the edges in this multigraph (by cutting wires, placing police at strategic drop-off points, or paying street urchins to ‘lose’ messages) to separate it into multiple components. Since his country is currently having an economic crisis, the agent wants to remove as few edges as possible to accomplish this task.

More formally, a *cut* partitions the nodes of G into two nonempty subsets. The *size* of the cut is the number of *crossing edges*, which have one endpoint in each subset. Finally, a *minimum* cut in G is a cut with the smallest number of crossing edges. The same graph may have several minimum cuts.



A multigraph whose minimum cut has three edges.

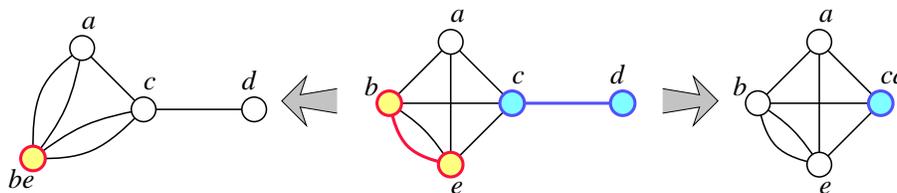
This problem has a long history. The classical deterministic algorithms for this problem rely on *network flow* techniques, which are discussed in Chapter 26 of CLRS. The fastest such algorithms run in $O(n^3)$ time and are quite complex and difficult to understand (unless you’re already familiar with network flows). Here I’ll describe a relatively simple randomized algorithm published by David Karger², how was a Ph.D. student at Stanford at the time.

Karger’s algorithm uses a primitive operation called *collapsing an edge*. Suppose u and v are vertices that are connected by an edge in some multigraph G . To collapse the edge $\{u, v\}$, we create a new node called uv , replace any edge of the form u, w or v, w with a new edge uv, w , and

¹A multigraph allows multiple edges between the same pair of nodes. Everything in this lecture could be rephrased in terms of simple graphs where every edge has a non-negative weight, but this would make the algorithms and analysis slightly more complicated.

²D. R. Karger*. Random sampling in cut, flow, and network design problems. Proc. 25th STOC, 648–657, 1994.

then delete the original vertices u and v . Equivalently, collapsing the edge shrinks the edge down to nothing, pulling the two endpoints together. The new collapsed graph is denoted $G/\{u, v\}$. We don't allow self-loops in our multigraphs; if there are multiple edges between u and v , collapsing any one of them deletes them all.



A graph G and two collapsed graphs $G/\{b, e\}$ and $G/\{c, d\}$.

I won't describe how to actually implement collapsing an edge—it will be a homework exercise later in the course—but it can be done in $O(n)$ time. Let's just accept collapsing as a black box subroutine for now.

The correctness of our algorithms will eventually boil down to the following simple observation: For any cut in $G/\{u, v\}$, there is a cut in G with exactly the same number of crossing edges. In fact, in some sense, the 'same' edges form the cut in both graphs. The converse is not necessarily true, however. For example, in the picture above, the original graph G has a cut of size 1, but the collapsed graph $G/\{c, d\}$ does not.

This simple observation has two immediate but important consequences. First, collapsing an edge cannot decrease the minimum cut size. More importantly, collapsing an edge increases the minimum cut size if and only if that edge is part of *every* minimum cut.

5.2 Blindly Guessing

Let's start with an algorithm that tries to *guess* the minimum cut by randomly collapsing edges until the graph has only two vertices left.

```

GUESSMINCUT( $G$ ):
  for  $i \leftarrow n$  downto 2
    pick a random edge  $e$  in  $G$ 
     $G \leftarrow G/e$ 
  return the only cut in  $G$ 

```

Since each collapse takes $O(n)$ time, this algorithm runs in $O(n^2)$ time. Our earlier observations imply that as long as we never collapse an edge that lies in every minimum cut, our algorithm will actually guess correctly. But how likely is that?

Suppose G has only one minimum cut—if it actually has more than one, just pick your favorite—and this cut has size k . Every vertex of G must lie on at least k edges; otherwise, we could separate that vertex from the rest of the graph with an even smaller cut. Thus, the number of incident vertex-edge pairs is at least kn . Since every edge is incident to exactly two vertices, G must have at least $kn/2$ edges. That implies that if we pick an edge in G uniformly at random, the probability of picking an edge in the minimum cut is at most $2/n$. In other words, the probability that we don't screw up on the very first step is at least $1 - 2/n$.

Once we've collapsed the first random edge, the rest of the algorithm proceeds recursively (with independent random choices) on the remaining $(n - 1)$ -node graph. So the overall probability that

GUESSMINCUT returns the true minimum cut is given by the following recurrence:

$$P(n) \geq \frac{n-2}{n} \cdot P(n-1).$$

The base case for this recurrence is $P(2) = 1$. We can immediately expand this recurrence into a product, most of whose factors cancel out immediately.

$$P(n) \geq \prod_{i=3}^n \frac{i-2}{i} = \frac{\prod_{i=3}^n (i-2)}{\prod_{i=3}^n i} = \frac{\prod_{i=1}^{n-2} i}{\prod_{i=3}^n i} = \boxed{\frac{2}{n(n-1)}}$$

5.3 Blindly Guessing Over and Over

That's not very good. Fortunately, there's a simple method for increasing our chances of finding the minimum cut: run the guessing algorithm many times and return the smallest guess. Randomized algorithms folks like to call this idea *amplification*.

```

KARGERMINCUT(G):
  mink ← ∞
  for i ← 1 to N
    X ← GUESSMINCUT(G)
    if |X| < mink
      mink ← |X|
      minX ← X
  return minX

```

Both the running time and the probability of success will depend on the number of iterations N , which we haven't specified yet.

First let's figure out the probability that KARGERMINCUT returns the actual minimum cut. The only way for the algorithm to return the wrong answer is if GUESSMINCUT fails N times in a row. Since each guess is independent, our probability of success is at least

$$1 - \left(1 - \frac{2}{n(n-1)}\right)^N.$$

We can simplify this using one of the most important (and easy) inequalities known to mankind:

$$1 - x \leq e^{-x}$$

So our success probability is at least

$$1 - e^{-2N/n(n-1)}.$$

By making N larger, we can make this probability arbitrarily close to 1, but never equal to 1. In particular, if we set $N = c \binom{n}{2} \ln n$ for some constant c , then KARGERMINCUT is correct with probability at least

$$1 - e^{-c \ln n} = 1 - \frac{1}{n^c}.$$

When the failure probability is a polynomial fraction, we say that the algorithm is correct *with high probability*. Thus, KARGERMINCUT computes the minimum cut of any n -node graph in $O(n^4 \log n)$ time.

If we make the number of iterations even larger, say $N = n^2(n - 1)/2$, the success probability becomes $1 - e^{-n}$. When the failure probability is exponentially small like this, we say that the algorithm is correct with *very high probability*. In practice, very high probability is usually overkill; high probability is enough. (Remember, there is a small but non-zero probability that your computer will transform itself into a kitten before your program is finished.)

5.4 Not-So-Blindly Guessing

The $O(n^4 \log n)$ running time is actually comparable to some of the simpler flow-based algorithms, but it's nothing to get excited about. But we can improve our guessing algorithm, and thus decrease the number of iterations in the outer loop, by observing that *as the graph shrinks, the probability of collapsing an edge in the minimum cut increases*. At first the probability is quite small, only $2/n$, but near the end of execution, when the graph has only three vertices, we have a $2/3$ chance of screwing up!

A simple technique for working around this increasing probability of error was developed by David Karger and Cliff Stein.³ Their idea is to group the first several random collapses a 'safe' phase, so that the cumulative probability of screwing up is small—less than $1/2$, say—and a 'dangerous' phase, which is much more likely to screw up.

The safe phase shrinks the graph from n nodes to $n/\sqrt{2}$ nodes,⁴ using a sequence of $n - n/\sqrt{2}$ random collapses. Following our earlier analysis, the probability that any of these safe collapses touches the minimum cut is at most

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

Now, to get around the danger of the dangerous phase, we use amplification. Instead of running through the dangerous phase one, we run it twice and keep the best of the two answers. And of course, we treat the dangerous phase recursively, so we actually obtain a larger binary recursion tree, which gets wider and wider as we get closer to the base case, instead of a single path. More formally, the algorithm looks like this:

$\overline{\text{CONTRACT}(G, m):}$ for $i \leftarrow n$ downto m pick a random edge e in G $G \leftarrow G/e$ return G	$\overline{\text{BETTERGUESS}(G):}$ $X_1 \leftarrow \text{BETTERGUESS}(\text{CONTRACT}(G, n/\sqrt{2}))$ $X_2 \leftarrow \text{BETTERGUESS}(\text{CONTRACT}(G, n/\sqrt{2}))$ return $\min\{X_1, X_2\}$
---	--

This might look like we're just doing the same thing twice, but remember that CONTRACT (and thus BETTERGUESS) is randomized. Each call to CONTRACT contracts a different random set of edges; X_1 and X_2 are almost always different cuts.

³D. R. Karger* and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. Proc. 25th STOC, 757–765, 1993. Yes, *that* Cliff Stein.

⁴Strictly speaking, I should say $\lfloor n/\sqrt{2} \rfloor$, because otherwise, we'd have an irrational number of nodes! But this only adds some floors and ceilings to our recurrences, which we know that we can remove with domain transformations, so I'll just take them out now.

BETTERGUESS correctly returns the minimum cut unless *both* of the first two lines gives the wrong result. A single call to BETTERGUESS(CONTRACT($G, n/\sqrt{2}$)) gives the correct answer if none of the min cut edges are CONTRACTED and if the recursive BETTERGUESS is correct. So we have the following recurrence for the probability of success for an n -node graph:

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

I don't know how to derive it from scratch, but we can easily prove by induction that $P(n) = 1/\lg n$ is a solution for this recurrence. The base case is $P(2) = 1 = 1/\lg 2$.

$$\begin{aligned} P(n) &\geq P\left(\frac{n}{\sqrt{2}}\right) - \frac{1}{4}P\left(\frac{n}{\sqrt{2}}\right)^2 \\ &= \frac{1}{\lg \frac{n}{\sqrt{2}}} - \frac{1}{4\lg^2 \frac{n}{\sqrt{2}}} \\ &= \frac{1}{\lg n - \frac{1}{2}} - \frac{1}{4(\lg n - \frac{1}{2})^2} \\ &= \frac{4\lg n - 3}{4\lg^2 n - 4\lg n + 1} \\ &= \frac{1}{\lg n} + \frac{1 - \frac{1}{\lg n}}{4\lg^2 n - 4\lg n + 1} \\ &> \frac{1}{\lg n} \checkmark \end{aligned}$$

The last step used the fact that $n > 2$; otherwise, that ugly fraction we threw out might be zero or negative.

For the running time, we get a simple recurrence that is easily solved using the Master theorem.

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

So all this splitting and recursing has slowed down the guessing algorithm slightly, but the probability of failure is *exponentially* smaller!

Now if we call BETTERGUESS $N = c(\lg n)(\ln n)$ times, for some constant c , the overall probability of success is

$$1 - \left(1 - \frac{1}{\lg n}\right)^{c(\lg n)(\ln n)} \geq 1 - e^{-c \ln n} = 1 - \frac{1}{n^c}.$$

In other words, we now have an algorithm that computes the minimum cut with high probability in only $\boxed{O(n^2 \log^3 n)}$ time!