15-451/651: Design & Analysis of Algorithms
Lecture #4 (draft!!) September 5, 2013

Today's menu:

- Randomized Algorithms that may make mistakes (though rarely).
- The Minimum Cut problem.
- A simple, fast randomized algorithm for minimum cut.

---

1 Monte Carlo versus Las Vegas

In Lecture #2, you saw Quick-Select, an algorithm for selection that always gives the right answer, but the running time is a random variable. There we showed that the *expected* running time was linear. In fact you may have seen (randomized) QuickSort in the past: there again the output was always correct — always a sorted array of the input numbers. Just the runtime was a random variable, and this was $O(n \log n)$ in expectation (and also with high probability).

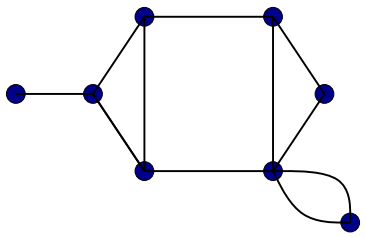
Today's randomized algorithm will be slightly different. It will always run in a given time bound, but may give incorrect answers some of the time. However, the probability of getting a correct answer will be at least (say) 3/4.

- Randomized algorithms which always output the correct answer, and whose runtimes are random variables, are called **Las Vegas** algs.
- Randomized algorithms which always terminate in given time bound, but output the correct answer with at least some probability (say with 3/4 prob.) are called **Monte Carlo** algs.

We are going to now look at a Monte Carlo randomized algorithm for the Minimum Cut problem.

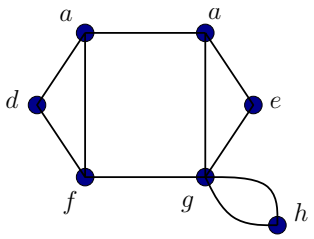
2 The Minimum Cut Problem

In this lecture we consider a undirected unweighted multigraph $G = (V, E)$. Let the number of vertices (aka vertices) be $|V| = n$. Number of edges is $|E| = m$.(In the rest of the lecture, whenever we say "graph" we really mean "multigraph".) E.g., in the multigraph below there are $n = 8$ nodes, $m = 11$ edges.



Given a multigraph G, a *cut* is a set of edges whose removal splits the graph into at least two connected components. A *minimum cut* is a cut of minimum size — i.e., with the minimum number of edges in it. For instance, say this graph represents a network, and we want to know how "robust" it is in the sense of the the minimum number of links whose failure causes the network to become disconnected. The minimum cut problem is to find a cut of minimum size.

In the multigraph above, the minimum cut has size 1, it consists of the single edge (c, d). If we consider the example below, what is the size of the minimum cut? What are the minimum cuts in this graph?



Answer: The minimum cut size is 2. There are 4 different min-cuts:

- $\{da, df\}$ separating d from the rest of the graph
- $\{eb, eg\}$ separating e from the rest of the graph
- the two edges between g and h, separating h from the rest of the graph
- $\{ab, fg\}$ separating $\{a, d, f\}$ from $\{b, e, g, h\}$

Fact 1 *The minimum cut has size at most the minimum degree of any node in the graph.*

Proof: The edges touching any node v can be deleted to separate v from the rest of the graph, and hence form a cut. The number of edges is the degree of v. Hence the minimum cut has size at most the minimum degree. ■

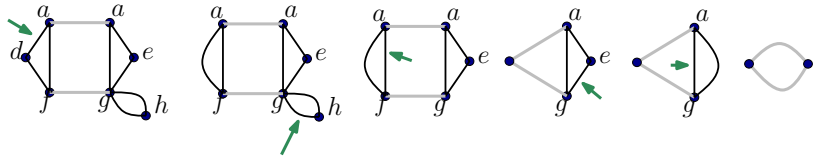
2.1 The Algorithm

Here's a really simple randomized algorithm: [due to D. Karger]

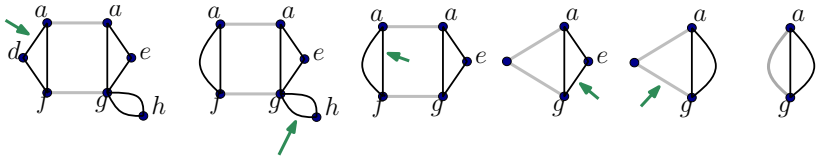
The Simple Algorithm (view #1)

1. Pick an edge (x, y) uniformly at random in current graph G.
2. **Contract** the edge, keeping multi-edges, but removing self-loops. (I.e., if there were edges (x, v) and (y, v), we keep both of them.)
3. If there are more than 2 nodes, go back to 1. Else, output the edges remaining as your cut.

Let's do an example on the graph: a mincut is marked in light grey and the randomly chosen edges are indicated by green arrows.



This was a successful run, where we actually output a minimum cut. Here's an unsuccessful run where we output a non-minimum cut (with 3 edges).



Easy algorithm. How well does it do? Clearly it doesn't always find the minimum cut...

Theorem 2 *The Simple Algorithm outputs a minimum cut with probability at least*

$$\frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \geq 1/n^2.$$

Hmm. This seems like (and is!) a tiny probability of being correct. (For a 1000 node graph, we are claiming a 1-in-million chance of being correct!) On the other hand, out of the 2^m possible subsets of edges in an n-vertex graph, this simple algorithm zones in on a minimum cut with probability $1/n^2$ — pretty awesome, actually! And we will soon show how to make the probability a lot better.

But first, let's prove this theorem.

2.2 The Proof of Theorem 2

There may be many minimum cuts in G. Fix some minimum cut C. Let it have k edges. We want to show this cut survives until the end.

Claim 3 *So long as none of edges in cut C have been selected, C remains a cut.*

Proof: We only contract edges, so things on left side of C stay on left side, and things on right side stay on right side. ■

During the course of the algorithm, does the min cut size go down or up?

Claim 4 ("Cuts Now were Cuts Earlier") *Any cut in the new graph is also a cut in the original graph.*

Proof: Why? Just think of undoing the last operation: all this does is split a node. This also means the min cut size can't go down. ■

Claim 5 ("Many Edges") *When have $n-i$ nodes remaining (for some $i \geq 0$), there are at least $(n-i)k/2$ edges in the current graph. (Remember, k is the size of the minimum cut in the original graph.)*

Proof: By Claim 4, the current minimum cut value is at least k. Hence by Fact 1, the degree of each node is at least the min-cut size, and hence at least k. Finally, each edge contributes one to the degree of each of its endpoints, so the total number of edges is the sum of the vertex degrees, divided by 2.

So, number of edges in the current graph is at least $k(n-i)/2$. ■

The point of Claim 5 is this: we want the graph to have a lot of edges at alll times, so the probability that the random edge chosen hits the cut C is small. And hence there's a good chance that we will not kill our favorite cut.

So ... Recall: k is the size of the min cut, and we've fixed some min cut C. What is the probability that C survives the first round?

$$\begin{aligned}
&\Pr[C \text{ not killed in round } \#1] \\
&= 1 - \Pr[C \text{ is hit by a random edge in round } \#1] \\
&= 1 - \frac{k}{\#\text{edges}} \\
&\geq 1 - \frac{k}{nk/2} \\
&= 1 - \frac{2}{n} = \frac{n-2}{n}.
\end{aligned}$$

Good. Now suppose C has survived so far, and there are now $n - i$ vertices. Exactly the same reasoning shows that

$$\begin{aligned}
&\Pr[C \text{ not killed in round } \#(i+1)] \\
&= 1 - \Pr[C \text{ is hit by a random edge in round } \#(i+1)] \\
&= 1 - \frac{k}{\#(\text{edges in round } i+1)} \\
&\geq 1 - \frac{k}{(n-i)k/2} \\
&= 1 - \frac{2}{n-i} = \frac{n-i-2}{n-i}.
\end{aligned}$$

So, the probability that cut C survives all $n - 2$ rounds until we are down to 2 vertices:

$$\begin{aligned}
&\prod_{i=0}^{n-3} \Pr[C \text{ survived round } \#(i+1)] \\
&\geq \prod_{i=0}^{n-3} \frac{n-i-2}{n-i} \\
&= \frac{(n-2)}{n} \times \frac{(n-3)}{(n-1)} \times \frac{(n-4)}{(n-2)} \times \ldots \times \frac{1}{3} \\
&= \frac{2}{n(n-1)}.
\end{aligned}$$

Since this is at least $1/n^2$, this finishes the proof of Theorem 2.

Exercise: Prove that if there are q different minimum cuts $C_1, C_2, ..., C_q$ in the graph, then the probability that our algorithm outputs a minimum cut is at least $\frac{2q}{n(n-1)}$. Note that you don't need the cuts to be *disjoint*, just distinct — i.e., $C_i \neq C_j$ for all $i \neq j$.

Exercise: Prove that any graph has at most $\binom{n}{2}$ minimum cuts. (Hint: probabilities can never be more than 1.)

Exercise: Show a family of graphs $G_2, G_4, \ldots$, one for every $n \geq 2$, such that each graph G_n has $\binom{n}{2}$ minimum cuts? (This shows the result proved in the previous exercise is tight.)

Exercise: Give a (multi)graph with a unique minimum cut C, such that the probability that our algorithm outputs C is $O(1/n^2)$.

2.3 The Running Time

OK, good. Also, while we're at it, how fast can we run this algorithm?

We need to assume something about the representation of the graph now. Let's assume that the graph is given in the form of adjacency lists. And that you can pick a random edge of the graph in constant time. (We may revisit this assumption later in the course.)

Then you should show that each round of the algorithm (where a round is "pick a random edge and contract it") can be implemented in $O(n)$ time. (How?) This gives us a runtime of $O(n^2)$ overall.

Or here's an equivalent formulation. Recall Kruskal's algorithm for finding a *minimum-weight spanning tree* of a graph; you would have seen it in 15-122 and 15-210. (Or in equivalent courses.)

The Simple Algorithm (view #2)

1. Randomly rank the edges. (flip coins in advance) Think of these as "weights" on the edges.
2. Use Kruskal's Algorithm to find minimum spanning tree (i.e., start with lightest edge, then put in next lightest that connects two different components, etc.)
3. Remove the last ("heaviest") edge added. This gives you the two components.

Exercise: Prove that this algorithm is equivalent to the view #1.

Since Kruskal's algorithm can be implemented in $O(m \log m)$ time, (or $O(n^2)$ time), we can run our algorithm in time

$$R(n) = \min(O(m \log m), O(n^2)).$$

3 Boosting the Success Probability

But still the probability of success is really lame! Like $1/n^2$. And to top that off, it's kinda weird: we don't know if we failed — to find that out we would need to figure out whether a given cut is a minimum cut or not. Which seems no easier than finding one. :(

However, here's the boost the success probability: Simple — just *try, try again*!

Run the above algorithm M times, independently.
Output the smallest cut found in these M runs.

The runtime of this algorithm is clearly $M \times R(n)$. What about the success probability?

Well, we are happy if one of these M runs found a minimum cut. In particular, we are happy if one of the M runs found the min-cut C. So if we fail, *all M runs must have failed to find C*. What is the chance of that?

$$\begin{aligned}
&\Pr[\text{ all } M \text{ runs of simple algorithm failed }] \\
&= \Pr[1^{st} \text{ run failed}] \times \Pr[2^{nd} \text{ run failed}] \times \cdots \Pr[M^{th} \text{ run failed}] \\
&\leq (1 - \tfrac{1}{n^2}) \times (1 - \tfrac{1}{n^2}) \times \cdots \times (1 - \tfrac{1}{n^2}) \\
&= (1 - \tfrac{1}{n^2})^M \qquad (\star)
\end{aligned}$$

And now we can use one of the most useful inequalities ever:

$$\boxed{(1 + x) \leq e^x \qquad \text{for all } x \in \mathbb{R}.}$$

So the failure probability ($\star$) is at most

$$\left(1 - \frac{1}{n^2}\right)^M \leq e^{-M/n^2}.$$

And so if we repeat $M = n^2 \ln n$ times, the failure probability is at most $e^{-\ln n} = 1/n$. If you think this is not enough, increase M to $100n^2 \ln n$ runs to get the failure probability to $e^{-100 \ln n} = 1/n^{100}$. Note: this is a tiny *failure* probability, early we had a tiny *success* probability.

> This is a useful trick to remember — if you have a Monte-Carlo algorithm with a small success probability, you can usually repeat the algorithm independently many times and reduce the failure probability.

Great! So today's result – a randomized (Monte-Carlo) algorithm to find the minimum cut in a graph with

- running time $M \cdot R(n) \leq O(mn^2 \log^2 n)$, and
- failure probability $1/n$.

4 More Speed? You Got It.

We might not get to this part in lecture — let's see how to speed up things a little more. I mean, the algorithm is still really simple (you could explain it to your grandmum), but she might complain about the running time of $O(mn^2 \log^2 n)$.

So here's a better speed-up due to D. Karger and C. Stein (the "S" in "CLRS").

Claim 6 *The earlier steps of the Algorithm are less risky than later steps.*

Why? See, the chance that we kill the cut C in the first step is at most $2/n$. But at the last step, we might have essentially a $2/3$ chance of killing the cut C.

Question: what is the probability that C still survives when we have t nodes left?

Answer: Doing the calculations like above, it is at most

$$\frac{t(t-1)}{n(n-1)}.$$

(Sanity check. After 2 nodes left, this should be $2/(n(n-1))$, to match our previous calculations. And it is.)

Question: At what point is our chance of having killed C already about 50/50?

Answer: When we have about $t = n/\sqrt{2}$ vertices left. By that time, our success probability is about:

$$\frac{(n/\sqrt{2}) \cdot (n/\sqrt{2} - 1)}{n(n-1)} \approx \frac{1}{2} \qquad (\star\star)$$

Hence the failure probability is about 50% as well.

So, this suggests a speedup:

Fast-Mincut(G)

1. Let $n \leftarrow$ number of nodes in G.
   If $n \leq 8$, find the min-cut by brute-force.
2. Run the simple algorithm until $t = n/\sqrt{2}$ nodes left. (I.e., pick random edge and contract, until $n/\sqrt{2}$ nodes left.) Call this graph H.
3. Recurse independently on H *twice*. I.e.,
   $C_1 \leftarrow$ **Fast-Mincut**(H)
   $C_2 \leftarrow$ **Fast-Mincut**(H)
   return smaller of the two.

Question: what is running time?

Answer: The recurrence is $T(n) = O(n^2) + 2T(n/\sqrt{2}) = O(n^2 \log n)$. E.g., use the stack-of-bricks argument or the master theorem.

Question: What is the probability that mincut C survives?

Answer: It is at least $\Omega(\frac{1}{\log n})$. Why?

Let P_n be the probability the cut C survives in a graph with n nodes. Let $n' = n/\sqrt{2}$. Then

$$\begin{aligned}
P_n &= \Pr[C \text{ survives the first few contractions}] \times \Pr[C \text{ survives one of two recursive calls}] \\
&\geq \frac{1}{2} \cdot \Pr[C \text{ survives one of two recursive calls}] \\
&= \frac{1}{2} \cdot (1 - \Pr[C \text{ killed in both recursive calls}] \\
&= \frac{1}{2} \cdot (1 - (1 - P_{n'})^2) \\
&= \frac{1}{2} \cdot (2P_{n'} - (P_{n'})^2) \\
&= P_{n'} - (1/2)(P_{n'})^2.
\end{aligned}$$

Exercise: Verify (by induction) that $P_n > 1/(c \log n)$ for some constant c.

Great! So the running time is slightly more than the simple algorithm (by a factor of $\log n$), but the success probability is exponentially larger ($\Omega(1/\log n)$ rather than $\Omega(1/n^2)$! We can again repeat this algorithm $O(\log^2 n)$ times to get the failure probability down to $1/n$.

Exercise: We were a little loose in equation ($\star\star$). To make the above completely precise, we should really stop when H has $t = \lceil \frac{n}{\sqrt{2}} + 1 \rceil$ nodes remaining. Show that this does not change the running time or the success probability except by constant factors.

To summarize today's (truly) final result – a randomized (Monte-Carlo) algorithm to find the minimum cut in a graph with

- running time $O(n^2 \log^3 n)$, and
- failure probability $1/n$.