CSE 347 Lecture 20

Randomized Algorithms III

1. Problem: global Min-Cut
2. Karger's Contraction Algorithm
3. Amplifying correctness

So far, we've seen randomized algorithms that relaxed the notion of worst-case running time to expected running time or a high-probability worst-case running time bound. It is also possible to consider algorithms that are (only) correct with high probability. As with the running time, the motivation is that randomized algorithms of this sort are faster and/or simpler than deterministic algorithms.

Many people, upon first hearing about algorithms that may make mistakes, are troubled by this. But, consider that computing hardware itself is not 100% reliable. If we can tolerate a one-in-a-million chance of our machine failing, we should be willing to tolerate it in our algorithms, too.

The Global Min-Cut Problem

Recall that a cut in a graph $G = (V, E)$ is a partition of the vertices V into two nonempty sets A & B.

Now, we suppose we are given an undirected (and unweighted) graph G. We wish to find a cut of G with the fewest edges crossing between A and B.

We previously saw how algorithms for Max-Flow solved the Minimum s-t cut problem, which is different
1. it refers to a weighted, directed graph G
2. the objective value is the total weight of edges crossing from A to B, not just the number of edges
3. it requires $s \in A$ and $t \in B$ (so the cut separates s&t).

Nevertheless, it is possible to use a subroutine for Minimum s-t cut to solve Global Min-Cut. How?

Given our undirected graph G, we construct a directed graph G' on the same vertex set such that for each undirected edge $(u,v) \in E$, we include both directed edges (u,v) and $(v,u) \in E'$ each with weight 1. Now, we put s equal to an arbitrary vertex, and iterate over the other $|V|-1$ possible choices for t. For each pair (s,t), we find a Min s-t Cut in G', and return the cut of minimum cost among these. Note that the cost of a cut (A,B) in G' is equal to the number of undirected edges that cross from A to B in G, so the cuts have the same value in G and G'. Moreover, in the min. cut of G, A^*, B^*, s must appear on one side — call this A — and the other side is nonempty, so there is some t^* in the other side, B. Thus, when our algorithm searches

for a minimum s-t^* cut, it returns some cut of G of cost at most that of A^*, B^* — since A^*, B^* is a cut of minimum cost, the cut (A, B) we find must be of the same cost. Indeed, there cannot exist a cut of smaller cost for any choice of t, so we return a cut of this cost.

Karger's Contraction Algorithm

We already have an algorithm for computing Global Min-Cut deterministically, so again, randomization is not fundamentally changing what we can compute. (This seems to be true in general for algorithms, but it is known that randomization is fundamentally necessary in distributed computing and cryptography.) But, our algorithm relies on the complicated apparatus of computing augmenting paths in residual graphs. By contrast here is a simple direct algorithm for Global Min-Cut.

The basic operation is that we pick an edge at random and contract it: that is, we replace the endpoints u & v with a single node w, and add an edge to w for each edge incident to u or v, unless it crosses between u and v. We'll let this new graph be a multigraph, so that for example there was a common vertex with an edge to both u and v, there will be two edges to w; in general, there may be any number of parallel edges. We will repeat this, iteratively reducing the # of vertices by one,

until only two vertices remain. The sets of vertices that were collapsed to form these "supernodes" are now the cut we return. In summary:

Input Undirected graph $G=(V,E)$

Initialize a set $S(v)=\{v\}$ for each $v \in V$.
Repeat $|V|-2$ times:
  Choose (u,v) from the remaining edges uniformly at rand.
  Delete all edges between $S(u)$ and $S(v)$.
  Put $S(u), S(v) \leftarrow S(u) \cup S(v)$.
Return the sets $S(u), S(v)$ such that $S(u) \neq S(v)$

Amazing! Does it really work? Well, sort of...

Theorem: Karger's Contraction Algorithm returns a global min. cut with probability at least $1/\binom{|V|}{2}$.

That is, there is a very small but noticeable chance that the algorithm actually works. Nevertheless, we'll see a fairly general technique for converting such algorithms into algorithms that almost always are correct later.

Proof: Let (A^*, B^*) be some global min. cut of G, and suppose that it has cost k, i.e., the set F of edges crossing from A^* to B^* has size k. The algorithm fails to return (A^*, B^*) if it ever chooses an edge in F, since then it places two vertices that should be on opposite sides of the cut in the same side. Actually, this is an "only if" as well: after $|V|-2$ iterations, if we never contract an

edge of F, we only combine sets $S(u)$ and $S(v)$ for $u,v \in A^*$ or $u,v \in B^*$, so each $S(w) \subseteq A^*$ or $S(w) \subseteq B^*$; since only two sets remain and A^* and B^* are disjoint, these must be the two sets.

So, we need to argue that there are many edges <u>not</u> in F, so that we are unlikely to pick an edge in F on each iteration. Here is the key observation: since (A^*, B^*) is a min. cut, <u>every other</u> cut (A', B') must have cost at least k; in particular, the single-vertex cuts $(v, V \setminus \{v\})$ for every $v \in V$ must have at least k edges crossing, i.e., every $v \in V$ has degree at least k. Thus G has at least $\frac{k|V|}{2}$ edges (why? $\sum_{v \in V} \deg(v)$ counts each edge twice, once per endpoint, so $2|E| = \sum_{v \in V} \deg(v) \geq \sum_{v \in V} k = k \cdot |V|$.) More generally, after we have collapsed several vertices into a set S, <u>each</u> such $(S, V \setminus S)$ is a cut that again must have k edges crossing. So, after the ith iteration, each of our $|V|-i$ "supernodes" still have degree at least k, so the graph must have at least $\frac{k(|V|-i)}{2}$ edges after iteration i. Notice, as long as we never pick an edge of F, F always has precisely k edges. So:

$\Pr[\text{don't pick } e \in F] = \Pr[\text{don't pick } e \in F \text{ on iteration } 1] \cdot$

$\Pr[\text{don't pick } e \in F \text{ on } i=2 \mid \text{don't pick on } i=1]$

$\vdots$

$\Pr[\text{don't pick on } i=|V|-2 \mid \text{don't pick on } i=1,2,\ldots,|V|-3]$

Where, for each i, $\Pr[\text{don't pick on } i \mid \text{don't pick on } 1,2,\ldots,i-1] \geq 1 - \frac{2k}{k(|V|-i+1)}$

(Since $\Pr[\text{pick on } i^{\text{th}} \text{ iter} \mid \text{don't pick on } 1,2,\dots,i-1] \le \frac{2k}{k(|V|-i+1)}$, we just showed.) So, $\Pr[\text{don't pick } e \in F] \ge \left(1-\frac{2}{|V|}\right)\left(1-\frac{2}{|V|-1}\right)\cdots\left(1-\frac{2}{3}\right)$

$$= \left(\frac{|V|-2}{|V|}\right)\left(\frac{|V|-1-2}{|V|-1}\right)\cdots\left(\frac{4-2}{4}\right)\left(\frac{3-2}{3}\right)$$

denominators cancel two terms later...

$$\frac{|V|-3}{|V|-1}\cdot\frac{|V|-4}{|V|-2}\cdot\frac{2}{4}\cdot\frac{1}{3}$$

i.e., we get $\frac{2\cdot 1}{|V|\cdot(|V|-1)} = \frac{1}{\binom{|V|}{2}}$ ▨

Amplifying Correctness

The algorithm is unlikely to work each time we run it, but suppose we run it ℓ times and return the best cut we see among these ℓ. Then, if we ever obtain a min. cut on any of these ℓ runs, we will return a min. cut, and the algorithm is correct. Thus, the probability that the algorithm is incorrect is at most $\left(1-\frac{1}{\binom{|V|}{2}}\right)^{\ell}$. So, for any $\delta > 0$, if we repeat $\ell = \binom{|V|}{2}\ln\frac{1}{\delta}$ times, then using $1-x \le e^{-x}$, $\left(1-\frac{1}{\binom{|V|}{2}}\right)^{\ell} \le e^{-\ell/\binom{|V|}{2}} = e^{-\ln 1/\delta} = \delta$. That is: the algorithm's output is then correct with probability at least $1-\delta$.

Notice, the same trick works for any optimization problem: a small probability ε of success can be amplified to a large probability $1-\delta$ by repeating $\frac{1}{\varepsilon}\ln\frac{1}{\delta}$ times. For decision (yes/no) problems, we can similarly convert an algorithm that is slightly better than a random guess (correct with probability $\frac{1}{2}+\varepsilon$) into one that is correct with probability $1-\delta$ by repeating it $\frac{4}{\varepsilon^2}\ln\frac{1}{\delta}$ times and taking a majority vote of the answers. Each run is a coin

toss of bias $p \geq \frac{1}{2}+\varepsilon$, so the probability that fewer than $(1-\varepsilon)\left(\frac{1}{2}+\varepsilon\right)\ell = \left(\frac{1}{2}+\varepsilon\left(\frac{1}{2}-\varepsilon\right)\right)\ell > \frac{\ell}{2}$ of the ℓ trials are correct is at most $e^{-\frac{1}{2}\varepsilon^2\left(\frac{1}{2}+\varepsilon\right)\ell} = e^{-\frac{1}{2}\varepsilon^2\left(\frac{1}{2}+\varepsilon\right)\frac{4}{\varepsilon^2}\ln\frac{1}{\delta}} = \delta^{1+2\varepsilon} < \delta$ by the Chernoff bound. Thus, the vote obtains the correct answer with probability at least $1-\delta$.