DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

The plan today is to discuss the ‘other’ style of graph algorithms mentioned last time: bottom up contractions. That is, we simply group together vertices until the answer becomes apparent.

This algorithm is easiest introduced through the minimum cut problem, which can be easiest described as removing the fewest number of edges so that the graph becomes disconnected.

A set of such edges is known as a cut. However, formally cuts are not parameterized as edges, but as vertex subsets. That is, we parameter a cut by a subset of vertices $S \subseteq V$. This subset can be viewed as one of the connected components formed by removing the edges. And given this subset, it’s clear what edges we need to remove: the ones incident to $S$. This set is $E(S, V \setminus S)$, and the size of the cut given by $S$ is

$$\partial(S) = |E(S, V \setminus S)|.$$

The minimum cut problem is then finding $S$ that minimizes $|E(S, V \setminus S)|$. We will analysis perhaps the most classical contraction based algorithm [KS96]:

1. Repeatedly contract a random edge.

2. Until two vertices remain, at that point, output the cut given by the vertices that became one of these two vertices.

Note that this contraction process naturally generates multi-edges: such duplicates skew the probability of edges being picked, and it’s crucial that we handle them properly.

This algorithm is simple to phrase, but difficult to both analyze and implement efficiently. We first consider its correctness, since the running time also depends on its success probabilities.

1 Correctness

First, note that this is a randomized algorithm. In general, we assume all the randomness we generate are independent: this means in most cases, we can just multiply together the probabilities of success.

Let $S^*$ be some minimum cut. If contracted edge not in $E(S^*, V \setminus S^*)$ ($S^*$: minimum cut), mincut in resulting graph same as in original.
So it suffices to bound the probability of hitting such an edge when \( n' \) vertices remain. If the minimum cut has size \( k \), then each vertex’s degree is at least \( k \) because \( \{u\} \) forms a cut. So the total number of edges is at least \( \frac{n'k}{2} \).

Thus, the probability of hitting an edge \( i \in E(S^*, V \setminus S^*) \) is at most

\[
\frac{k}{(n'k)/2} \leq 1/(n'/2) = 2/n'.
\]

Telescoping this from \( n' = n \) to \( n' = 2 \):

\[
\prod_{n'=3}^{n} \left(1 - \frac{2}{n'}\right) = \left(\frac{1}{3}\right) \left(\frac{2}{5}\right) \cdots \left(\frac{n-3}{n-1}\right) \left(\frac{n-2}{n}\right) = \frac{2}{(n-1)n}.
\]

Thus, we terminate with \( S^* \) and \( V \setminus S^* \) with probability at least \( 1/n^2 \).

## Efficiency

This success probability means the algorithm needs to run a total of \( n^2 \) iterations for a constant probability of success, or \( O(n^2 \log n) \) iterations to succeed with high probability. In each of the iterations, we can perform a contraction in \( O(n) \) time, for a total cost of \( O(n^3) \). Thus, if we take all the math at face value, we obtain a running time of \( O(n^5 \log n) \).

These factors of \( \log n \) usually only make the first order calculation of running time more complicated. So I like to handwave them away by saying ‘about \( n^5 \)’ instead: this is fine as long as I compose an algorithm with itself a constant number of times.

We can do MUCH better. First, note that we are still picking a edge randomly: just that some of the edges do not benefit from being contracted because their endpoints are already the same. But should that happen, we simply delete the edge from our pool of candidates, which happens at most once.

So up to the cost of union find, which is \( \alpha(n) \) (at most 5 in practice), we can run the entire contraction process in \( O(m) \) time. This leads to an overall runtime of about \( n^2 m \), which is still as large as \( n^4 \) on dense graphs.

We can do even better by analyzing when are the failures likely to happen: if we start with \( n \) vertices, and end with \( n/k \) vertices, the success probability is actually

\[
\frac{(n/k - 1) (n/k - 2)}{n (n-1)} \geq \frac{1}{10k^2}.
\]

So the majority of the failures are expected to happen at a much smaller, or later, part of the progress.

Specifically, if we plug in \( k = n^{0.1} \), that is, run the algorithm until the problem size is \( n^{0.9} \), the success probability is at least \( \Omega(n^{-0.2}) \). This implies an iteration count of about \( n^2 \), which combined with the \( n^4 \) bound above gives that the total runtime (for succeeding with high probability) is about

\[
n^{0.2} \cdot (n^{0.9})^4 \approx n^{3.8},
\]
which is less than the \(n^4\) we had before.

Whenever this happens, we should repeat it more times. In general, if we set our reduction factor to \(k\), we get a running time of the form of

\[
T(n) = k^2 \cdot T\left(\frac{n}{k}\right) + n^2,
\]

which if we set \(k\) to \(n^\delta\) for \(\delta \to 0\), solves to about \(O(n^{2+\epsilon})\) for any \(\epsilon > 0\). That is, the algorithm is about as good as it can be for dense graphs.

### 3 More Musings

I’m not convinced that the above algorithm, when combined with proper data structures, can’t be made to run in almost linear time. That is, about \(m^{1+\omega(1)}\) for any constant.

**Open Problem.** Give an \(O(m^{1+\omega(1)})\) time contraction based algorithm for computing minimum cuts.

This result may be folklore: it’s possible it wasn’t written down due to the subsequent \(O(m \log^3 n)\) result [Kar00].

The contraction algorithm also has structural implications. Observe that \(S^*\) could be ANY minimum cut. It implies:

**Theorem 1.** For any value \(\alpha > 1\), there are at most \(n^{2\alpha}\) different cuts whose values are at most \(\alpha\) times the minimum cut.

Note in particular this implies that there are at most \(n^2\) distinct minimum cuts in a graph. This is tight for a cycle.

The proof is basically repeating the above argument, but for a larger cut:

**Proof.** Consider any cut \(S^*\) of size at most \(\alpha\) times the minimum.

The probability of hitting an edge in this cut is at most

\[
\frac{\alpha}{n'/2} = 2^{\frac{\alpha}{n'}}.
\]

Telescoping ‘shifted’ by \(2\alpha\) terms gives a product of at most \(n^{2\alpha}\).

There are two additional issues to address:

- \(2\alpha\) needs to be an integer. Fix: Gamma function or lose an additive 1.

- \(2\alpha/n' > 1\) when \(n' \leq 2\alpha\). Fix: switch to picking a random subset at that point. Extra term of \(2^{(2\alpha)}\) absorbed by numerators.

Finally, we remark that shortest path may also be solvable via contractions. We simply contract together all vertices on level \(i\) of the BFS tree into a single vertex. Contractions can only decrease distance, and resulting graph has a path from \(s\) to \(t\). This, of course, is very much musing. However, there is a good chance that there are more algorithms of this type to be discovered/analyzed.
References
