Advanced Heuristic and Approximation Algorithms Grenoble-INP (ORCO)

Lecture 2

Lecturer: Alantha Newman *October 9, 2017*

1 Algorithms for Min-Cut

Given an undirected, unweighted multigraph $G = (V, E)$, the (global) *minimum cut problem* (min-cut) is to find a proper, nonempty subset of vertices $S \subset V$ such that $|E(S, V \setminus S)|$ is minimized. An edge $ij \in E$ belongs to $E(S, V \setminus S)$ if one endpoint belongs to S and the other endpoint belongs to the complement of S, namely $\bar{S} = V \setminus S$.

Another variant of min-cut is known as the *s-t-min-cut*. Here, we are given two specified vertices $s, t \in V$ and we are asked to find S such that $s \in S$ and $t \notin S$ for which (again) $|E(S, V \setminus S)|$ is minimized. Min-cut can be reduced to s-t-min-cut by solving the latter problem for every choice of s and t. A reduction in the other direction is not known.

1.1 Using s-t-Max-Flow

For a given graph $G = (V, E)$, we use n and m to denote $|V|$ (the number of vertices) and $|E|$ (the number of edges), respectively. Applying the famous Max-Flow/Min-Cut Theorem, the s-t-min-cut can be solved via a reduction to s-t-max-flow. An instance of s-t-Max-Flow can be solved in time $\tilde{O}(nm)$. (We use $\tilde{O}$ to hide log factors.) Therefore, a naive bound on the running time required to solve min-cut is $\tilde{O}(n^3 m)$. In fact, it was shown that the solutions for all s-t-min-cuts (over all choices of s and t) can be found in the same running time as one max-flow computation. (For example, see [HO94]. For further relevant citations, see page 4 of [KS96].)

1.2 Without Max-Flow

Can we solve min-cut *without* using max-flow? Nagamochi and Ibaraki gave an algorithm that (deterministically) finds in $O(m)$ time an edge that does not belong to any min-cut [NI92]. (If all edges belong to some min-cut, they presumably gave a procedure to find a min-cut.) Then they used the following *contraction* procedure to modify the (multi) graph $G = (V, E)$. Informally, this procedure contracts an edge ij by merging vertices i and j to a single vertex and deleting the resulting self-loop(s). Define $\delta(i) := \{ik \in E \mid k \in V, k \neq i\}$.

CONTRACT(G, $ij \in E$)

1. Merge vertices i and j to form a single vertex u.
2. Let $V' = \{V \setminus \{i, j\}\} \cup \{u\}$.
3. Let $E' = \{E \setminus \{\delta(i), \delta(j)\}\}$.
4. For each $k \neq i, j$:
   (a) If $ik \in E$, add edge uk to E'.
   (b) If $jk \in E$, add edge uk to E'.
5. Output (multigraph) $G' = (V', E')$.

It should be clear that if an edge ij does not belong to a minimum cut, then contracting vertices

i and j does not increase the value of the minimum cut. It also the case that even if an edge ij *does* belong to a minimum cut, the contraction procedure never results in a smaller minimum cut.

Fact 1. *For any edge $e \in E$, the graph G' output by* Contract*(G,e) does not have a smaller minimum cut than G.*

The running time of Contract(G,e) is $O(n)$. Therefore, the running time of Nagamochi and Ibaraki's algorithm is roughly $O(mn)$, which is not faster than reducing min-cut to max-flow.

2 Random Contraction

Karger observed that for a multigraph $G = (V, E)$, an edge chosen uniformly at random from E has a low probability of belonging to a particular minimum cut. So rather than investing $O(m)$ time to deterministically search for such an edge, he presented a beautiful algorithm for min-cut, in which the key idea is to contract random edges [Kar93].

Random-Contract(G)

1. Iterate $n - 2$ times:
   (a) Pick edge ij uniformly at random from E.
   (b) $G \leftarrow$Contract(G, ij).

2.1 Analysis

Let $G = (V, E)$ be a multigraph with minimum degree d. Consider a fixed minimum cut $(S, \bar{S})$ with size C, i.e. $C = |E(S, \bar{S})|$. We will say that the algorithm Random-Contract(G) *succeeds* if it never contracts any edge in $E(S, \bar{S})$. The following fact should be clear.

Fact 2. *Algorithm* Random-Contract(G) *outputs cut $(S, \bar{S})$ if and only if no edge in $E(S, \bar{S})$ is ever contracted.*

Recall Fact 1, which states that the minimum cut never decreases when an edge is contracted. Also note that each time an edge is contracted, the resulting graph has one less vertex. We observe the following:

1. $C \leq d$.
2. $|E| \geq \frac{nd}{2} \geq \frac{nC}{2}$.

Suppose we choose an edge ij uniformly at random from E. What is the probability that ij belongs to $E(S, \bar{S})$?

$$\Pr[\text{edge } ij \text{ belongs to } E(S, \bar{S})] \;=\; \frac{C}{|E|} \;\leq\; \frac{C}{\frac{nd}{2}} \;\leq\; \frac{C}{\frac{Cn}{2}} \;=\; \frac{2}{n}.$$

So,

$$\Pr[\text{edge } ij \text{ does } \textit{not} \text{ belong to } E(S, \bar{S})] \;\geq\; 1 - \frac{2}{n}.$$

Thus, the probability that we *never* choose an edge from $E(S, \bar{S})$ during each of the $n-2$ iterations of the algorithm is:

$$\begin{aligned}
\Pr[\text{cut } (S,\bar{S}) \text{ is output}] &= \Pr[\text{no edge of } E(S,\bar{S}) \text{ is chosen}] \\
&\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{n-(n-3)}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{3}{5}\right)\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&= \frac{1}{\binom{n}{2}}.
\end{aligned}$$

Since each minimum cut in G is output by the algorithm RANDOM-CONTRACT(G) with probability at *at least* $\frac{1}{\binom{n}{2}}$, we have the following corollary.

Corollary 3. *A graph contains at most* $\binom{n}{2}$ *minimum cuts.*

Exercise 1. *Find a graph with exactly* $\binom{n}{2}$ *minimum cuts.*

Exercise 2. *True or False: For a graph* $G = (V, E)$*, let* $S_1 \subset V$ *and* $S_2 \subset V$ *denote two distinct, nonempty subsets of vertices, each corresponding to a minimum cut of* G*. Then each of these cuts is equally likely to be output by the routine* RANDOM-CONTRACT(G)*.*

The probability of finding a particular minimum cut might seem small when compared to deterministic algorithms with which you are likely familiar; Kruskal's algorithm and Dijkstra's algorithm output a minimum spanning tree and a shortest path, respectively, with probability 1. However, we can apply a technique commonly used to *amplify* the success of a randomized algorithm.

Claim 4. *Run the algorithm* RANDOM-CONTRACT(G) *for* $10 \cdot \binom{n}{2} \log n$ *times. Then a fixed minimum cut* $(S, \bar{S})$ *is output with probability at least* $1 - \frac{1}{n^{10}}$*.*

Proof. Probability that we do not obtain the minimum cut $(S, \bar{S})$ after running RANDOM-CONTRACT(G) $k = 10 \cdot \binom{n}{2} \log n$ times is:

$$\begin{aligned}
\left(1-\frac{1}{\binom{n}{2}}\right)^k &= \left(\left(1-\frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}}\right)^{10\log n} \\
&\leq \left(\frac{1}{e}\right)^{10\log n} \\
&= \frac{1}{n^{10}}.
\end{aligned}$$

(Note that $\log n$ is the natural log.) □

A bound in the form of $1 - \frac{1}{poly(n)}$ is called *with high probability* (w.h.p. for short).

2.2 Running Time Analysis

The algorithm RANDOM-CONTRACT consists of $O(n)$ edge contractions, and each execution of CONTRACT(G,e) can be implemented in time $O(n)$. Thus, the total running time of a single execution of RANDOM-CONTRACT is $O(n^2)$. To obtain the high probability bound, the running time is $\tilde{O}(n^4)$. However,

an implementation of RANDOM-CONTRACT(G) can actually be run in $\tilde{O}(m)$ time, resulting in a total running time of $\tilde{O}(mn^2)$.

Observe that choosing $n-2$ edges to contract during an execution of algorithm RANDOM-CONTRACT is the same as first choosing a random permutation of the edges and then contracting edges in that order. Thus, we can consider the first $\frac{m}{2}$ edges and check how many vertices would result if all these edges have already been contracted. This can be performed in $O(m)$ time using depth-first-search to compute the number of connected components induced by these m edges. If there are fewer than two components, we only need to consider the first $\frac{m}{2}$ edges as candidate edges to be contracted. If there are more than two components, then we consider the next $\frac{m}{4}$ edges, etc. Thus, in at most $\log m$ steps, we find the point in the permutation where the last edge would be contracted by the algorithm.

Exercise 3. *True or false: An execution of* RANDOM-CONTRACT *can actually be run in* $\tilde{O}(n)$ *time, because it is enough to consider only the first* $n-2$ *edges in the permutation.*

3 Min-Cut in Time $\tilde{O}(n^2)$

We now show how to find a minimum cut with high probability in time $\tilde{O}(n^2)$, which is faster than max-flow. This algorithm is due to Karger and Stein [KS96]. The implementation described previously runs RANDOM-CONTRACT $\tilde{O}(n^2)$ times. However, note that in an execution of RANDOM-CONTRACT, after the the first $O(n)$ edge contractions, it is quite unlikely that an edge from a fixed minimum cut will be contracted. In particular, after the first $O(n - \frac{n}{\sqrt{2}})$ edge contractions, the probability that we have not contracted an edge belonging to a particular minimum cut is about $\frac{1}{2}$.

Thus, we can avoid doing "extra" work. Roughly speaking, we do the following. We set $\ell = \frac{n}{\sqrt{2}}$. Then we make two copies of G and in each copy, we contract $n - \ell$ edges. (This means that each copy has ℓ vertices after the edge contractions are performed.) Then we recursively run this procedure on each of the two copies.

RECURSIVE-CONTRACT(G)

1. $n = |V|, \quad \ell = \frac{n}{\sqrt{2}}$.
2. If $n < 6$, RETURN (size of) minimum cut of G.
3. Else:
   (a) For $k = 1, 2$:
      i. $G_k = (V_k, E_k) \leftarrow G$.
      ii. Iterate $n - \ell$ times:
         A. Pick edge ij uniformly at random from E_k.
         B. $G_k \leftarrow$CONTRACT(G_k, ij).
   (b) RETURN min{RECURSIVE-CONTRACT(G_1), RECURSIVE-CONTRACT(G_2)}

Let $T(n)$ denote the running time of RECURSIVE-CONTRACT(G) when G has n vertices. Then $T(n)$ can be analyzed via the following recurrence relation.

$$T(n) \quad = \quad O(n^2) + 2 \cdot T(\frac{n}{\sqrt{2}}).$$

Claim 5. $T(n) = O(n^2 \log n)$.

Thus, the running time of RECURSIVE-CONTRACT is $\tilde{O}(n^2)$. It remains to show the following.

Claim 6. *The algorithm* RECURSIVE-CONTRACT(G) *returns a minimum cut with probability* $\frac{1}{\log n}$.

Proof sketch. Let $P(n)$ denote the probability that the algorithm succeeds on a graph G with n vertices. Then for one of the copies of G, the probability that it returns a minimum cut is $\frac{1}{2} \cdot P(n/\sqrt{2})$. So the probability that neither copy returns a minimum cut is:

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

And the probability that at least one copy returns a minimum cut is:

$$P(n) \quad = \quad 1 - (1 - \frac{1}{2} \cdot P(\frac{n}{\sqrt{2}}))^2.$$

The proof proceeds by induction to show that $P(n) \geq \frac{1}{\log n}$. (For details, see page 15 of [KS96].)

Since the probability of success is at least $\frac{1}{\log n}$, we can run RECURSIVE-CONTRACT $O(\log^2 n)$ times to find a minimum cut with probability at least $1 - \frac{1}{n}$.

Exercise 4. *True or false: Since the probability of success, i.e. returning a minimum cut, of* RECURSIVE-CONTRACT *is at least* $\frac{1}{\log n}$*, we can conclude that there are actually at most* $\log n$ *minimum cuts in a graph.*

References

[HO94] Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17(3):424–446, 1994.

[Kar93] David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30, 1993.

[KS96] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.