Principles of Distributed Computing 05/15, 2019

Lecture 11

Lecturer: Mohsen Ghaffari *Scribe:*

1 Minimum Cut

In this lecture, we explain a distributed algorithm that computes a $(2+\epsilon)$ approximation of the minimum cut, for any constant $\epsilon > 0$, in $O(k(D+\sqrt{n})\log^3 n)$ rounds. Here, k denotes the size of the minimum cut, i.e., the minimum number of edges whose removal disconnects the graph. The algorithm should present a nonempty subset of vertices S such that the number of edges connecting nodes of S to nodes of $V \setminus S$ is at most $(2+\epsilon)k$. As usual, we work in the standard synchronous message-passing model of distributed computing where the network is abstracted as an n-node connected graph with diameter D and per round each node can send $O(\log n)$ bits to each of its neighbors.

A number of remarks are in order:

- First, the k factor in the round complexity can be removed and replaced with $O(\log n)$, using a sampling idea which reduces the minimum cut to $O(\log n)$, while preserving the sizes of all cuts approximately.
- Second, the algorithm can be generalized to weighted graphs where each edge has a weight in $\{1, 2, \ldots, \text{poly}(n)\}$ and the size of a cut is the summation of the weights of the edges whose removal disconnects the graph. For this generalization, we simply view each edge of w as w many parallel edges.
- Third, the resulting $\tilde{O}(D+\sqrt{n})$ round complexity is nearly optimal, as a known lower bound shows that any distributed algorithm for (any non-trivial) approximation of the minimum cut requires at least $\tilde{\Omega}(D+\sqrt{n})$ rounds. See Ghaffari and Kuhn [MK13] for details of the above three points.
- Fourth, a $(1+\epsilon)$ approximation algorithms with a round complexity of $\tilde{O}(D+\sqrt{n})$ is known to a work of Nanongkai and Su [NS14]; we will not cover it in this lecture.
- Finally, in a recent breakthrough, Daga et al. [DHNS19] presented the first distributed algorithm that computes the exact minimum cut in a sublinear number of rounds, particularly in $\tilde{O}(n^{1-1/353}D^{1/353} + n^{1-1/706})$ rounds in unweighted graphs. More recently, Ghaffari and Nowicki [GN19] presented a different algorithm that improved the round complexity further to $\tilde{O}(n^{1-1/9}D^{1/9} + n^{1-1/18})$.

1.1 Sparse Certificates for Connectivity

Before presenting the algorithm for minimum cut approximation, first we introduce the concept of *sparse certificates.* In a very rough sense, this is a subgraph that maintains the size of the minimum cut while reducing the number of edges. Notice that any graph that has minimum-cut size at least k must have at least $nk/2$ edges. This is because each node in such a graph must have degree at least k, as otherwise removed the edges adjacent to that node would yield a cut with size less than k. A sparse certificate aims to reduce the number of the edges in any graph to something close to this threshold of $nk/2$, while maintaining the minimum cut size.

Definition 1. *For a graph $G = (V, E)$, we call a spanning subgraph $H = (V, E_H)$ a* sparse certificate of G for connectivity k' *if the following two conditions are satisfied:*

(1) For each nonempty subset $S \subset V$ of vertices, we have $cut_H(S, V \setminus S) \geq \min\{k', cut_G(S, V \setminus S)\}$. Here, $cut_H(S, V \setminus S)$ denotes the number of edges in H that have one endpoint in S and the other endpoint in $V \setminus S$. Similarly, $cut_G(S, V \setminus S)$ denotes the number of edges in G that have one endpoint in S and the other endpoint in $V \setminus S$.

(2) Subgraph H has at most $|nk'|$ edges.

Example: In a connected graph G, any spanning tree T is a sparse certificate for connectivity $k' = 1$. It clearly satisfies condition (2), because it has $n - 1$ edges, and it satisfies condition (1) as well because $Cut_T(S, V \setminus S) \geq 1$. More generally, if we do not assume G to be connected, any maximal spanning forest is a sparse certificate for connectivity $k' = 1$.

Computing a Sparse Certificate for Connectivity k' (Algorithm Outline): As the above example reveals, there is a simple and natural algorithm for computing a sparse certificate for connectivity k'. Here, we discuss the algorithm outline from a centralized viewpoint. We later discuss a distributed algorithm that implement this outline, efficiently. The algorithm works in k' iterations, as follows: In iteration i, compute a maximal spanning forest F_i of G and then update $G \leftarrow G \setminus F_i$. Here, $G \setminus F_i$ means we remove from G all edges of F_i. At the end of algorithm, the union $H = \bigcup_{i=1}^{k'} F_i$ of the forests computed in the k' iterations is our desired sparse certificate for connectivity k'.

Lemma 2. *In the above algorithm, $H = \bigcup_{i=1}^{k'} F_i$ is a sparse certificate for connectivity k'.*

Proof. Since the number of edges in each forest F_i is at most $n - 1$, graph H has at most $k(n - 1) \leq kn$ edges and thus it clearly satisfies condition (2). To see that it also satisfies condition (1), let us focus on an arbitrary cut $(S, V \setminus S)$. In each iteration i, so long as at least one edge of G remains in the cut $(S, V \setminus S)$, the forest F_i will include at least one edge from $(S, V \setminus S)$. Notice that it might also include more than edge. Hence, the number of edges of H in cut $(S, V \setminus S)$ is either at least k, or the algorithm exhausted all the edges of this cut and therefore the number is equal to $cut_G(S, V \setminus S)$. □

Distributed Implementation of the Above Outline: A naive idea for implementing the above outline would be to compute each F_i separately, in the remaining graph. However, this might be quite inefficient as the remaining graph might have a very large diameter. To go around this issue, we use some artificial edge weights to implement the outline. In particular, initially, we set the weight of each edge to be 1. Then, in each iteration i, we compute a minimum-weight spanning tree T_i and define F_i to be all edges of T_i that have weight 1. Then, instead of removing these edges from the graph, we simply set their weight to be ∞ (or realistically, just n^2). Computing each minimum-weight spanning tree T_i can be done in $O((D + \sqrt{n}) \log n)$ rounds, using the MST algorithm that we discussed in the previous lecture.

Lemma 3. *Consider the weighted version of G where we set the edge weight of $G \setminus (\cup_{j=1}^{i-1} F_i)$ to be 1 and the edge weights of $(\cup_{j=1}^{i-1} F_i)$ to be n^2. Then, the edges of weight 1 in the minimum-weight spanning tree T_i form a maximal spanning forest of the graph $G \setminus (\cup_{j=1}^{i-1} F_i)$.*

Proof Idea. Informally, T_i will try to include as many as possible of weight 1 edges before including any weight n^2 edges. Formalizing this into a proof is left as an exercise. □

1.2 Approximation Algorithm for Minimum Cut

We now describe the algorithm for computing a $(2 + \varepsilon)$ approximation of the minimum cut. For now, we assume that the value k of the minimum cut size is known. The algorithm will compute a subset $S \subset V$ such that $cut_G(S, V \setminus S) \leq (2 + \varepsilon)k$.

Intuitive Description of the Algorithm: First, we compute a sparse certificate H for connectivity $k' = (1+\varepsilon/10)k$. Notice that this can be done in $O(k(D+\sqrt{n})\log n)$ rounds, as we discussed above. Now, think about the auxiliary graph H' where we *contract* each edge of $G \setminus H$, that is, we unify the two endpoint nodes into one node. Notice two properties: (A) Edges of H' are exactly edges of H (while the nodes where contracted) and therefore H' has at most $nk' = (1+\varepsilon/10)nk$ edges. Moreover, (B) every cut of H' has size at least k, and every cut of H' that has size less than k' must have had size less than k' also in G. Hence, H' has minimum cut size k and any minimum cut of it corresponds to a minimum cut of G (and vice versa). Now, H' might be in one of the following two cases. In each case, we can handle the problem differently.

(1) Suppose that H' has at least $n/(1+\varepsilon/5)$ nodes. In this case, the average degree in H' is at most $\frac{2(1+\varepsilon/10)nk}{n/(1+\varepsilon/5)}$ which implies that H' has at least one node with degree at most $2k(1+\varepsilon/10)(1+\varepsilon/5) \leq (2+\varepsilon)k$. This node in H' is the result of contracting some edges of G, and is therefore some subset S of vertices of G (which were contracted together). We get that the number of edges from S to $V \setminus S$ is at most $(2+\varepsilon)k$, that is, S is our desired approximate minimum cut. Hence, if we are in the case that H' has at least $n/(1+\varepsilon/5)$ nodes, then the problem is easy.

(2) Suppose that H' has less than $n/(1+\varepsilon/5)$ nodes. In this case, it is not so clear to identify a small cut. However, we have managed to make some progress because the number of nodes has reduced to $n/(1+\varepsilon/5)$, while we have preserved the minimum cut. Then, we can recurse on this new graph H' by setting $G \leftarrow H'$, which has at most $n/(1+\varepsilon/5)$ vertices. Notice that after $O(\log_{1+\varepsilon/5} n)$ such recursions, the number of remaining nodes would drop to at most 2 and thus, there, we have found our approximate minimum cut.

Distributed Implementation The above intuitive description provides the outline of one level of recursion, in our algorithm. We now discuss how to implement it in a distributed setting. Suppose we are in a setting where we are working on a subgraph G' of G, identified by its edges. Initially, in the very first iteration, G' will be same as G. In each iteration, at the end of the iteration, if we are in case (2) and need to recurse, if we want to contract edges of $G \setminus H$, we would set $G' \leftarrow H$. Effectively, each connected component of $G \setminus G'$ corresponds to one node in our contracted structure.

The algorithm in this iteration of recursion works as follows: we set the weights of edges of $G \setminus G'$ to be 0 and weights of edges of G' to be 1. Then, we compute a sparse certificate H of G' for connectivity $k' = (1+\varepsilon/10)k$, as we described in the previous subsection (iteratively computing MSTs, taking one-weight edges as F_i, and increasing their weight to n^2). At the end of the sparse certificate computation, by the definition of a sparse certificate, subgraph H has a number of edges that is at most a k' factor of the number of connected components of $G \setminus G'$. Hence, we are virtually moving to a graph where all edges of $G \setminus H$ are contracted (weight 0) and no edge of H is contracted.

Then, each component of $G \setminus H$ computes the number of its edges in H, using the algorithm that we saw in the previous lecture where each component could compute the minimum of its edges — now, instead of minimum, we compute summation, but the algorithm is the same. This can be done in $O((D+\sqrt{n})\log n)$ rounds, as we saw last week. If for at least one of the components, the number of edges is at most $(2+\varepsilon)k$, then we have found our approximate minimum cut. Otherwise, we set $G' = H$ and recurse.

Final Remark — Removing the Assumption of Known k: In the above, we assumed that we know the value of k. In fact, the outline works perfectly fine if instead we know an estimation of k that is at least k and at most $k(1+\varepsilon/10)$ (Why? Explain what happens in the algorithm and how the calculations change). To remove the assumption of knowing this value, we simply run the algorithm for many estimates, which are powers of $(1+\varepsilon/10)$. That is, we run the algorithm for each estimate of the form $(1+\varepsilon/10)^i$ between 1 and n^2. Notice that there are only $O(\log_{1+\varepsilon/10} n)$ such powers, in this range. From each estimate, we get some cut (which might or might not be a small cut). The algorithm sets its output to be the smallest of these cuts. We know that one of these estimates $(1+\varepsilon/10)^i$ must be at least k and at most $k(1+\varepsilon/10)$. For that estimate, the algorithm outputs a cut that has size at most $(2+\varepsilon)k$. For other estimates, the cut size might be much larger, but since we output the minimum, we know that the output cut will have size at most $(2+\varepsilon)k$.

References

[DHNS19] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In *Proc. of the Symp. on Theory of Comp. (STOC)*, page arXiv:1904.04341, 2019.

[GN19] Mohsen Ghaffari and Krzysztof Nowicki. Faster algorithms for edge connectivity via random out contractions. *Manuscript*, 2019.

[MK13] M. Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 1–15, 2013.

[NS14] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 439–453, 2014.