Principles of Distributed Computing	05/08, 2019
Lecture 10	
Lecturer: Mohsen Ghaffari	*Scribe:*

1 Minimum Spanning Tree

In this lecture, we discuss a near-optimal distributed algorithm in the CONGEST model for the problem of computing a minimum spanning tree (MST). We note that, over the years, the problem of computing an MST has turned out to have a central role in distributed algorithms for network optimization problems, significantly more central than its role in the centralized algorithms domain. The upper and lower bound techniques for the MST problem are used frequently in solving other distributed network optimization problems.

A first-order summary of global network optimization problems Distributed algorithms for global network optimization problems have a long and rich history. A first-order summary of the state of the art is that, for many of the basic problems, including MST, Minimum Cut Approximation, Maximum Flow Approximation, and Shortest-Paths Approximations, the best-known upper bound is $\tilde{O}(D + \sqrt{n})$ rounds, where D denotes the network diameter. Furthermore, it is known that this round complexity is essentially the best-possible in general graphs, that is, there are graphs in which any (non-trivial approximation) algorithm for these problems requires $\tilde{\Omega}(D + \sqrt{n})$ rounds [PR99, Elk04, DSHK$^+$11].

1.1 MST: The Algorithm Outline

The algorithm we describe follows the outline of Boruvka's MST[NMN01] from 1926, though with some small changes. In particular, we have $O(\log n)$ phases. During these phases, we gradually grow a forest until we reach a spanning tree. We start with the trivial forest where each node forms its own component of the forest, that is, each node is one separate component in our partition of G.

In each phase, each component S_i will have a leader node $s_i \in S_i$, and moreover, the leader will know the size of its component. Each component S_i suggests a merge along the edge with exactly one endpoint in S_i that has the smallest weight among such edges. This is called the *minimum weight outgoing edge (MWOE)*. Recall from the second lecture (lemma 2.17) that all such edges belong to MST[1]. We soon explain how to compute these min-weight outgoing edges, one per part, using shortcuts. Let us for now continue with the high-level explanation of how to use these edges to merge parts.

Let N be the current number of connected components of the forest. If $N = 1$, we are done already. Otherwise, each component suggests one merge edge. Each edge might be suggested by two of its endpoints, so we have at least $N/2$ suggested edges in total. We add these edges to the forest and thus, effectively, merge the connected components at their two endpoints. Hence, the number of components shrinks to at most $N/2$. After $\log n$ iterations, the number of connected components is down to 1, which means we have reached a spanning tree.

What remains is to explain how to find the merge edges, and how to perform the merges. We first discuss the process of computing minimum-weight edges of one phase in $O(D + \sqrt{n})$ rounds. Then, we will discuss how to perform the merges in the same round complexity.

[1]This assumes that the edge-weights are unique, which is effectively without loss of generality, because we can append the identifier of the edge—composed of the identifiers of its two endpoints—to its weight in a manner that makes the edge weights unique, and guarantees that the MST according to the new weights is one of the MSTs according to the original weights.

1.2 Computing Min-Weight Outgoing Edges

Our objective is to let each node know the the minimum weight outgoing edge of its component. More concretely, let each node v set $c(v)$ to be the minimum-weight outgoing edge among edges incident to v. The objective is that each node $v \in S_i$ learns the weight of the minimum-weight outgoing edge among edges all edges incident on component S_i. Notice that node v can easily find $c(v)$ by first receiving from all neighbors the component leader IDs of their components and then only considering the smallest of those edges having the other endpoint in a different component. We handle the components in two categories of small and large, depending on whether the component has at least $\sqrt{n}$ vertices or not.

Small Components Consider a single small component S_i, which means this component has no more than $\sqrt{n}$ vertices. Then, in this component, each node v starts with its own smallest weight-outgoing edge and its weight $c(v)$. Then, we perform a convergecast (or simply minimum flooding) on the BFS tree of this component S_i. This convergecast goes from the leaves to the root, maintaining the minimum value seen, and thus eventually delivering the minimum-weight outgoing edge to the component leader. The information about this edge can be delivered to all nodes of the component by a broadcast from the root to the leaves.

Large Components There are at most $n/\sqrt{n} = \sqrt{n}$ large parts, as each of them has at least $\sqrt{n}$ vertices. Since the number of large components is relatively small, we can handle all these components by performing their communications on the BFS of the whole graph G, simultaneously. Using standard pipelining techniques[Pel00], we can compute the minimum-weight outgoing edges of all these $\sqrt{n}$ components in $O(D + \sqrt{n})$ rounds.

A more general treatment

The above division to small and large categories, and then the rules for where each of these should communicate, leads to an $O(D + \sqrt{n})$-round algorithm. This is nearly-optimal in the worst case. However, in many graphs families of interest, this would be quite far from desirable bounds. In the following, and mostly as side remarks, we introduce a graph-theoretic notion of *low-congestion shortcuts* and briefly outline how this notion leads to more efficient algorithms, as well as a simple and clean unification of many methods.

Definition 1 (Low-Congestin Shortcuts)**.** *Consider a graph $G = (V, E)$ and a partition of V into disjoint subsets $S_1, \ldots, S_N \subset V$, each inducing a connected subgraph $G[S_i]$. We define an α-congestion shortcut with dilation β to be a set of subgraphs $H_1, \ldots, H_N \subset G$, one for each set S_i, such that:*

(1) For each i, the diameter of the subgraph $G[S_i] + H_i$ is at most β.

(2) For each $e \in E$, the number of subgraphs $G[S_i] + H_i$ containing e is at most α.

Theorem 2. *Suppose that the graph family $\mathcal{G}$ is such that for each graph $G \in \mathcal{G}$, and any partition of G into vertex-disjoint connected subsets $S_1, \ldots, S_N$, we can find an α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leq K$. Here, K can be a function of the family $\mathcal{G}$, and it can depend on n and D. Moreover, we assume such a low-congestion shortcut can be found in $\tilde{O}(T)$ rounds.*

Then, there is a randomized distributed MST algorithm that computes an MST in $\tilde{O}(T) + O(\alpha \log n + \beta \log^2 n) = \tilde{O}(T + K)$ rounds, with high probability, in any graph from the family $\mathcal{G}$.

It is not hard to see that the above rule for small and large components can be used to infer that any graph has a α-congestion β-dilation shortcut such that $\max\{\alpha, \beta\} \leq D + \sqrt{n}$,

and thus leads to an $\tilde{O}(D+\sqrt{n})$-round MST algorithm for general graphs. There are a number of graph families where the question, and especially its graph-theoretic aspects, becomes much more interesting:

- For planar graphs and a few generalizations (e.g. bounded-genus graphs), it has been shown[MH16] that there always exists an α-congestion β-dilation shortcut such that $\max\{\alpha,\beta\} \leq \tilde{O}(D)$ and moreover, such a shortcut can be found in $\tilde{O}(D)$ rounds. This leads to an $\tilde{O}(D)$-round algorithm for MST in planar and near-planar graphs.
- In Erdős-Renyi random graph $G_{n,p}$, where each of the possible $\binom{n}{2}$ edges is included with probability $p \geq \Omega(\log n/n)$ (that is, above the connectivity threshold), it has been shown that there always exists a low-congestion shortcut with $\max\{\alpha,\beta\} \leq \text{poly}\log n$, and such a shortcut can be found in $2^{O(\sqrt{\log n \log\log n})}$ rounds. That leads to an $2^{O(\sqrt{\log n \log\log n})}$-round algorithm for MST in Erdős-Renyi random graphs. See [GKS17] for this result and extensions to much broader graph families (those with small random walk mixing time).

1.3 Back to Worst-Case Graphs, Merging Components

A small change in Boruvka's outline to have low-depth merges We restrict the merges to be *star* shapes, using a simple random coin idea: toss a random coin per component and then allow only merges centered on head-parts, each accepting incoming suggested merge-edges from tail-parts. The leader of this head-component becomes the leader of the merged new part. In exercise 2 of today's lecture, we see that, albeit this slightly slowed down probabilistic merging process, still after $O(\log n)$ phases, with high probability, we reach a tree.

What we need to compute for a merge We need to make all nodes learn, besides the minimum-weight outgoing edge of their component, two extra things: (1) the coin tossed by their component leader, (2) the ID of their new component leader, (3) the size of the new component. We next explain how to perform each of these steps in $O(D+\sqrt{n})$ rounds.

- Item (1)—which is to let each node know the coin toss of its component leader—is by a simple small change to the messages sent in computing the min-weight outgoing edge: now the message starting at the root also carries the random bit flipped by the leader.
- Item (2)—which is to let each node know its new component leader ID—is performed as follows: We define the component leader ID to be the leader of the center component of the merge, who had a head coin. This ID is already delivered to the physical endpoint of the merge edge in the tail part.

  Hence, within the tail component, all that we need to do is that one node knows the ID of the new leader and we want all nodes to have it. If the component was small, we can do it directly in $O(\sqrt{n})$ rounds, inside the component. For large components, which there are only at most $\sqrt{n}$ of them, we can broadcast the their new ID leader, which is known to the physical endpoint of their merge edge, to all nodes of the graph in $O(D+\sqrt{n})$ rounds.
- Item (3)—which is to let each node know the size of its component—can be performed similar to (2). First, in the center-of-merge head component, the physical endpoints of the merge can receive from their other endpoints the sizes of the merging tail components. Then, within this head component, we can compute the new component size by performing a simple converge-case, if the component is small, and by doing it through the global BFS tree, for large components. At the end, this information can be passed to all vertices of the new component, by first delivering it to all nodes of the head component, then passing it through the physical edges to the tail components, and then spreading it in the tail components.

References

[DSHK+11] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 363–372, 2011.

[Elk04] Michael Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 331–340, 2004.

[GKS17] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distrbuted MST and routing in almost mixing time. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2017.

[MH16] M. Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: Low-congestion shortcuts, MST, and min-cut. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2016.

[NMN01] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1):3–36, 2001.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PR99] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed MST construction. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 253–, 1999.