A Survey on Algorithms for Computing Distributed Minimum Spanning Tree

Sitong Li (Student ID:251021439)

Department of Computer Science, University of Western Ontario

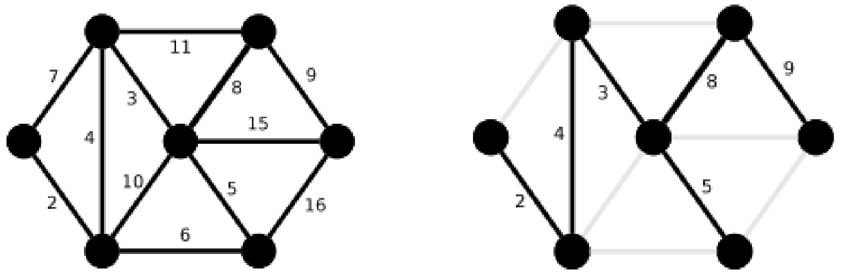
Abstract

The computation of a minimum-weight spanning tree (MST) over a network topology is a fundamental and classic problem in graph theory. In distributed systems, the need of computing MST occurs in the design and operation of communication networks, where weights could be thought as network parameters such as transmission delays, communication costs and so on. Therefore, it is desirable to compute MSTs thus with which the cost can be minimized or some profit can be maximized. Moreover, as a classic and representative problem in distributed computation, studying distributed MST is also of theoretical interest. Techniques, innovations and results are naturally extensible to other related problems such as Leader Election, Shortest Paths, etc.

In this report, we survey algorithms for the distributed minimum-weight spanning tree problem. More specifically, we mainly pay attention on five important algorithms in the literature. We describe their algorithmic procedures, identify the assumptions upon which each algorithm was built on. We then compare the message and time complexities between different methods. Finally, we conclude the report by pointing out potential future research directions as well as open questions which are to be answered by following-up researchers.

Keywords: Distributed Minimum Spanning Tree, GHS, GKP, KP, Elkin's Algorithm

Email address: sli2232@uwo.ca (Sitong Li (Student ID:251021439))



(a) Graph with edge weights (b) A minimum spanning tree

Figure 1: Minimum spanning tree example

1. Introduction

Given an undirected graph $G = (V, E)$ and a function w, where V, E and w are respectively the set of vertexes, the set of edges and $w : E \rightarrow \mathcal{R}^+$ assigns a positive weight for each edge, the MST problem asks to find a spanning tree over G with minimum summed weights. Figure 1 shows an example.

Sequential Algorithms. The study of minimum-weight spanning tree solutions dates back to Borúvka in 1926 [1] for efficient electrical coverage of Moravia. The algorithm works in a sequence of stages, named *Boruvka step*; It defines a forest F made of minimum-weight edge incident to each vertex in the graph G, then the reaming graph $G' = G\backslash F$ is treated as input for the next step. The algorithm terminates when all the nodes has been covered by a single tree, i.e., $|F| = 1$. Since each *Boruvka step* takes linear time, the number of vertexes is reduced to by at least a factor of 2, the whole procedure takes $O(|E|log|V|)$ time. The second method is due to Prim in 1957, and also Dijkstra in 1959. Starting from arbitrary vertex of G, the algorithm grows an MST noted as T sequentially by selecting an edge with the smallest weight from those edges connected to T. It stops when all vertexes in G are included in T. The runtime of Prim's algorithm is $O(|E|log|V|)$ or $O(|E|+|V|log|V|)$, depending on the specific choice of data structures used in implementation. Different from vertex centered decision making, Kruskal's algorithm makes greedy selection directly on edges: starting from the edge with the minimum weight, the algorithm iteratively finds a smallest weighted edge among the remaining, guaranteed that the no cycles would be incurred by adding it to the MST. Kruskal's algorithm runs at $O(|E|log|V|)$.

It is worth to mentioning that all MST algorithms are built on the following graph properties:

- **Cut property.** For any cut C of the graph, if the weight of an edge e in the cut-set of C is strictly smaller than the weights of all other edges of the cut-set of C, then this edge belongs to all MSTs of the graph.
- **Cycle property.** For any cycle C in the graph, if the weight of an edge e of C is larger than the individual weights of all other edges of C, then this edge cannot belong to an MST.

However, it is difficult to apply Prim and Kruskal's algorithms in the distributed message-passing model, primarily due to the following challenges:

- They require processing one node or vertex at a time, which makes it difficult to make them run in parallel, e.g. Kruskal's algorithm processes edges in turn, deciding whether to include the edge in the MST based on whether it would form a cycle with all previously chosen edges.
- They require processes to know the state of the whole graph, which is difficult to discover in the distributed message-passing model.

Distributed MST. Due to these difficulties, new techniques are needed for distributed MST in the message-passing model. The GHS algorithm by Gallager, Humblet and Spira [2] is the first algorithm of such. It bears similarity to Boruvka. The algorithm can construct an MST in both synchronous and asynchronous Message-passing models. It runs at $O(|V|\log|V|)$ communication rounds. This was soon enhanced to $O(|V|\log\log|V|)$ [3], later improved to $O(|V|\log*|V|)$ [4], and consequently to an existential optimal $O(|V|)$ [5]. Subsequent developments therefore try to identify more sensitive parameter in measuring the difficulty of the problem other than number of vertexes in the graph. Garay et al [7] proposed an algorithm that runs in sublinear time with respect to $|V|$, and linear to diameter of the network. He argued that the diameter is usually a more accurate parameter in describing the hardness of constructing an MST, thus it is desirable to devise algorithms with reduced complexity to such a metric. With this motivation, they developed an algorithm, noted as GKP [7], that uses two separate parts to carefully merge the fragments generated by controlled-GHS and then eliminate redundant edges. Subsequently, the first part of the GKP [7] is improved to $O(diameter(G) + \sqrt{|V|} \log *|V|)$ by following-up research using a fast distributed $k-$ Dominating Set construction procedure [8]. Kutten and Peleg [8] also claimed that their algorithm is close to the "neighborhood optimal" lower bound. Elkin [9] opened another line of research by proposing yet another parameter $\mu(G, w)$ to better capture the complexity of distributed MST. He argued his newly developed method could be remarkably faster than that of [8] in some but not all cases. Lotker et al. [10] considers an important special case where each node can directly send message to any other in the network, and proposed an algorithm that runs at $O(\log\log|V|)$.

Distributed MST is of fundamental importance, it is one of the central and most studied problems in network algorithms. The goal of distributed MST is to compute an MST in a distributed communication-based manner, at the same time striking to minimize two fundamental performance metrics in distributed algorithms, i.e. message and time complexities. In distributed systems, a processor is represented as a node in the graph and initially it only has local knowledge of its incident edges (represents communication channels and the cost of of using these channels for communication). Each processors starts with only local knowledge of the network topology, after an MST is constructed by a distributed algorithm, every processor should be aware which its incident edges belong to the MST and which not.

For the application side, it is obvious that a distributed MST can serve as an efficient communication backbone with which certain tasks such as broadcasting can be achieved with the minimum communication cost if the edge weight models the communication expense at each channel. In addition, distributed MST can be also be used for leader election, a fundamental problem in distributed computation. The Leader Election problem asks to select a node in the network as th leader. Naturally, if some rooted tree can be constructed, the root node can be recognized as the leader. Distributed MST constructs a special rooted tree with minimum summed edge weight, where the root can serve the leader without dissent.

In this article, we survey algorithms and bounds for the distributed MST problem. The rest of the article is organized as follows. In Section 2, we describe and compare different algorithms in detail. We then conclude the report by raising some cirtical questions for future research.

2. Algorithms and Comparisons

In this section, we present detailed description of five classic algorithms in the literature for distributed minimum-weight spanning tree problem, and finally give comparisons in terms of time and message complexities.

2.1. Notations and Definitions

Before describing the algorithms, it is necessary to introduce some notations and definitions used throughout this report, as listed in below:

- $G = (V, E)$ is the graphical abstraction of the network topology. V is the set of vertexcies and E is the set of edges. $\forall e \in E$, $w(e)$ represents the weight of edge e. We use n and m to denote the number of vertexes and edges, respectively, i.e., $n = |V|$, $m = |E|$.
- Besides n and m, other parameters could also be used to measure the "complexity" or "difficulty" of constructing an MST. In this report, we resort to $diameter(G)$ as the maximum number of hops between nodes in the network, i.e. diameter of G.

2.2. GHS

The GHS [2] is the first invented algorithm for distributed computation of spanning tree. We begin by review the assumptions made by GHS:

1. The network topology G is a connected undirected graph.
2. G has distinct finite weights assigned to each edge, though this assumption can be removed since one can simply appending each node's unique id to its adjacent edge weights or devise a consistent way for tie-breaking(e.g. by using a node's unique id). Indeed if both the ID of nodes and weights of edges are non-unique, no algorithm exists for computing MST in bounded rounds [6].
3. Each edge weight is initially known to its endpoint nodes.
4. Each node is initially in a quiescent state; It either spontaneously awakens or is awakened by receipt of any message from another node.
5. Messages are passed in both directions independently. They may arrive after an unpredictable but finite delay, but without error.
6. Messages are delivered in FIFO order at each edge.
7. Each message has size $O(\log n)$.

We note that such a model is also named as $\mathcal{CONGEST}$ in the literature. If the last assumption is relaxed, i.e., messages are permitted to be unbounded, the model is referred to as $\mathcal{LOCAL}$.

For convenience of presentation, GHS defines the following terms:

- Edge has three different types: A *branch* edge is an edge that has been determined to be part of the MST; A *rejected* edge is an edge that has been determined not to be part of the MST; The remaining are *basic* edges whose statuses are unsettled.

- A *fragment* F is a subgraph of G. Initially, there are n singleton fragments where each fragment contains exactly one node and zero edge. For each fragment F, $T(F)$ denotes the tree induced by the fragment. $level(F)$ is the level value of each fragment, initially 0. For each fragment F, the edge with one endpoint in F and another not in F whose weight is the smallest is called Minimum Weight Outgoing Edge (MWOE).

The synchronized GHS works in an level by level manner. The following pseudocode depicts the algorithm. Note that the description is essentially high-level; Unnecessary details on how to implement each step, either through *broadcasting*, *converge casting* or *consensus*, are ignored for simplicity.

Algorithm 1: GHS algorithm

```
Input: Network specification $G = (E, V)$ and $w$.
Output: Minimun spanning tree $T$
Before level 0: Each fragment will 1) find its MWOE 2) mark the
selected edge as brach 3) send message via branch node to notify node
on the other side 4) wait for response;
while There are more than one fragment on $G$ do
    Each fragment finds its MWOE;
    Merge two fragments appropriately if they share the same MWOE ;
    Each fragment increase its level by 1;
```

Time Complexity. It can be shown that each fragment at level l has at least 2^l nodes, so it takes $O(\log n)$ rounds to ensure there is one remaining fragment, i.e., the MST. Each round has complexity linear to the number of vertexes of G, so the overall complexity is $O(n \log n)$.

2.3. GKP

Observing that the network *diameter* could be a more sensitive parameter in measuring an algorithm's performance, Garay et al [7] proposed a sublinear time algorithm which runs at $O(diameter(G)+n^{0.614})$, arguing it as an improvement over previous best known $O(n)$ algorithm [5] since in many cases the network's diameter is significantly smaller than the number of vertexes. This algorithm is noted as GKP.

Additionally, the authors of GKP argued that their algorithm is "universally optimal" in the sense that this time complexity is ensured for all possible network instances.

The assumptions made by GKP include:

1. Each message has size $O(\log n)$. On the other hand, if message can be in unbounded size, it is trivial that an MST can be constructed in $O(diameter(G))$ by collecting the network topology in one node and then compute the MST locally.
2. A node may send at most one message on each edge each time step.
3. Edge weights are polynomial of n, so that each weight can be sent within one message.

Indeed, GKP may be regarded as a genuine enhancement of GHS, and is applicable wherever GHS can be used. The advantage of GKP over GHS is that it never result intermediate fragments with too large diameters, which could potentially slow down the termination due to expensive fragment merging. More specifically, each phase of GKP consists two parts,

- **Part 1.** Run `Controlled-GHS` for I phases. As depicted in Algorithm 2.

- **Part 2.** Use `Pipeline` method for edge elimination, illustrated in Algorithm 3.

Analysis of `Controlled-GHS` at phase I:

- The number of resultant fragments is at most $\frac{|V|}{2^I}$. This is because each phase will at least reduce the number of fragments by a half.

- For every fragment F, its diameter is at most 3^I. This is much due to the careful merging strategy as described by the `Controlled-GHS`, making that each new fragment's diameter is at most 3 times larger than its precedent.

Algorithm 2: Controlled-GHS

```
Function Controlled-GHS-Main():
    /* Stage 1                                                    */
    Execute GHS until every fragment $F$ has found its MWOE
    /* Stage 2:  break resulting trees into small trees
       with constant bounded size, then merge small ones          */
    Let $FF$ be the union of all fragments
    $MF \leftarrow \emptyset$
    foreach $F \in FF$ do
        $T \leftarrow tree(F)$
        $M(T) \leftarrow Small - Dom - Set(T)$
        $MF \leftarrow MF \cup M(T)$
    foreach $F \in FF - MF$ do
        $F' \leftarrow neighbor(FF \cap MF)$
        merge $F'$ and $F$
    return

Function Small-Dom-Set($T$):
    Mark each node of $T$ using its depth number $\tilde{L}(v) = 0, 1, 2$
    Let $R$ be the unmarked subtree of $T$
    Compute Maximum Independent Set (MIS) $Q$ on $R$
    $M \leftarrow Q \cup \tilde{L}_1$
    return $M$
```

Next is Part 2 `Pipeline`, which aims to eliminate potential cycles produced by the `Controlled-GHS`. The input `Pipeline` procedure is the fragment graph of phase I, noted $\tilde{F}_I$, which is composed by the all inter-fragment edges and their endpoints.

Analysis of this pipeline algorithm shows that it runs at time complexity $diameter(G) + \frac{n}{2^I}$.

Overall complexity of GKP: Since when the algorithm terminates, only one fragment remains, thus $3^I = \frac{n}{2^I}$, which means $I = \frac{\ln n}{\ln 6} = 0.613...$ So the overall complexity of GKP is $O(diameter(G)+n^{0.613+\epsilon}) \in O(diameter(G)+n^{0.614})$.

Algorithm 3: Edge Elimination by Pipeline

```
Function Pipeline():
    /* compute breath-first search tree B on G                    */
    $B \leftarrow bfs(G)$
    Let $r(B)$ be the root of $B$
    In the execution, each node $v$ on the tree maintains a set $Q$ of all
    the inter-fragment edges it knows of., stored in non-decreasing
    order by edge weights.
    From leaf $v$ starts to send edges upward, noted as pulse 0. Internal
    nodes start to send upward when itit has received message from all
    its children.
    At each pulse $i$, the node sends up to its parent in the tree the
    lightest edge $e$ in $Q$ that has not been sent up in the previous
    pulses up to $i - 1$.
    The root $r(B)$ computes locally set $S$ of $N - 1$ edges appearing in
    MST of the fragment graph $\tilde{F}$, from the edges from its children.
    The root $r(B)$ broadcasts over $B$ the resulting set of $N - 1$
    inter-fragment edges to all nodes in $G$.
```

2.4. KP

Kutten and Peleg [8] proposed a fast distributed construction of small $k-$ dominating sets and applied it the distributed MST problem. The main observation is that the performance of constructing an MST is mainly undermined by the "congestion" problem, i.e., the need to send the description of many edges. Thus, reducing the number of intermediate fragments as well the number of nodes at each fragment may further improve the time complexity.

Kutten and Peleg achieved this by introducing the concept of $k-$ Dominating Set, which is essentially a generalization of Dominating Set where a dominator dominates only its adjacent vertexes. A dominator in $k-$Dominating Set, however, can dominate vertexes that are k hops within its range. Naturally, each $k-$dominator v can be interpreted as the center of a "cluster" which include all nodes dominated by v.

The success of KP is built upon the following lemma:

Lemma 2.1. *For the* `Pipeline` *procedure, suppose it is running on an input consists of N fragments of the MST, its runtime is bounded by $O(N + diameter(G))$.*

In GKP, after running the `Controlled-GHS` for I iterations, it is obvious that the number of fragments could be $N = \frac{n}{2^I}$; where the optimal I then solved by letting $3^I = \frac{n}{2^I}$, then $N = n^{\frac{\ln 3}{\ln 6}} = n^{0.613\dots}$ If, however, one can reduce N, the overall complexity might be improved, given complexity of the first part still remains lower then procedure `Pipeline`.

KP achieved such a goal by proposing a fast $k-$ Dominating Set construction algorithm. Their final algorithm shares exactly the same Part 2 as GKP, but results only $N = \sqrt{n}$ fragments as the input for `Pipeline` by letting $k = \sqrt{n}$.

Notably, a lower bound on the time complexity of constructing an MST has been shown to be $\Omega(\frac{\sqrt{n}}{\log n} + diameter(D))$, indicating that the algorithm KP might be very close to the optimal.

2.5. Elkin's Protocol

Different from previous protocols that primarily focus on optimizing the construction of MSTs based on parameter $diameter(G)$, Elkin [9] noticed that another parameter called *MST-radius* may more accurately capture the complexity of the problem. The MST-radius, which is denoted $\mu(G, w)$, is a function of the graph topology as well as the edge weights, which could be interpreted as the maximum radius each vertex has to examine to see if any of its edges is in the MST.

Based on this notation, Elkin proposed a new protocol that constructs an MST in $\tilde{O}(\mu(G,w)+\sqrt{n})$ time. Here $\tilde{O}$ means a polylogarithm factor, i.e. $\log^k(n)$ with $k \geq 0$, is ignored. He further noted that although the new protocol does not guarantee to be universally better than [8], there are many cases it runs significantly faster. Indeed, on some graphs, the ratio between diameter and MST-radius could be as large as $\Theta(n)$, thus the new protocol runs faster than KP [8] by a factor of $\Omega(\sqrt{n})$. However, to correctly terminate the protocol, it requires $\mu(G,w)$ to be given as part of the input, which may be an unrealistic assumption in practice.

The message complexity of Elkin's algorithm is $O(m+n^{1.5})$.

2.6. Lotker et al

The next algorithm we describe is by Lotker et al [10]. In contrast to the aforementioned approaches, Lotker et al [10] instead developed a fast $O(\log\log n)$ algorithm specifically for a kind of network with constant diameter of 1. Networks of such kind deserves special attention since they represent a simple model for *overlay* network where even non-adjacent nodes can still communicate directly. Further, if the size of each message is relaxed from $O(\log n)$ to $\Theta(n^\epsilon)$ for some $\epsilon > 0$, the time complexity is $O(\log \frac{1}{\epsilon})$.

The message complexity this new algorithm is $O(n^2 \log n)$.

2.7. Comparisons

We summarize the differences of the above algorithms in terms of performance measurement parameter, model assumption and time-message complexity. As shown by the tables in below.

Table 1: Time Complexity Measurement Parameter

Algorithm	Reference	Measurement
GHS	Gallager et al. [2]	n
GKP	Garay et al. [7]	$diameter(G)$ and n
KP	Kutten and Peleg [8]	$diameter(G)$ and n
Elkin	Elkin [9]	$\mu(G,w)$ and n
Lotker	Lotker et al. [10]	n

Table 2: Built on Model Assumptions

Algorithm	Reference	Model
GHS	Gallager et al. [2]	$\mathcal{CONGEST}$
GKP	Garay et al. [7]	$\mathcal{CONGEST}$
KP	Kutten and Peleg [8]	$\mathcal{CONGEST}$
Elkin	Elkin [9]	$\mathcal{CONGEST}$ or $\mathcal{LOCAL}$
Lotker	Lotker et al. [10]	$diameter(G) = 1$ $\mathcal{CONGEST}$

Table 3: Time and Message Complexities

Algorithm	Reference	Time	Message
GHS	Gallager et al. [2]	$O(n \log n)$	$O(m + n \log n)$
GKP	Garay et al. [7]	$O(diameter(G) + n^{0.614})$	$O(m + n^{1.614})$
KP	Kutten and Peleg [8]	$O(diameter(G) + \sqrt{n} \log *n)$	$O(m + n^{1.5})$
Elkin	Elkin [9]	$\tilde{O}(diameter(G) + \sqrt{n})$	$\tilde{O}(m)$
Lotker	Lotker et al. [10]	$O(\log n \log n)$	$O(n^2 \log n)$

3. Conclusions

We have presented a brief survey of the development of algorithms for solving the distributed minimum-weight spanning tree problem. In contrast to the sequential version of problem, we have seen that algorithms for the distributed system may employ demarcated assumptions about the computation model, parameter of interest, and other practical concerns. This causes different algorithmic innovations from each line of research being developed independently.

On the other hand, it has been shown that the existing algorithms in the literature are already close to optimal in general. We contemplate that further research will continue to identify specific cases of interest, and advance the state-of-the-art in that respect. It remains a challenging task on how to further close or reduce the tight gap between theoretical lower and upper bounds in the most general case. It is also interesting to develop algorithms that are simultaneously time and message optimal.

References

[1] J. Nešetřil, E. Milková, H. Nešetřilová, Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history, Discrete mathematics 233 (2001) 3–36.

[2] R. G. Gallager, P. A. Humblet, P. M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Transactions on Programming Languages and systems (TOPLAS) 5 (1983) 66–77.

[3] F. Chin, H. Ting, An almost linear time and o (nlogn+ e) messages distributed algorithm for minimum-weight spanning trees, in: Foundations of Computer Science, 1985., 26th Annual Symposium on, IEEE, pp. 257–266.

[4] E. Gafni, Improvements in the time complexity of two message-optimal election algorithms, in: Proceedings of the fourth annual ACM symposium on Principles of distributed computing, ACM, pp. 175–185.

[5] B. Awerbuch, Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems, in: Proceedings of the nineteenth annual ACM symposium on Theory of computing, ACM, pp. 230–240.

[6] D. Angluin, Local and global properties in networks of processors, in: Proceedings of the twelfth annual ACM symposium on Theory of computing, ACM, pp. 82–93.

[7] J. A. Garay, S. Kutten, D. Peleg, A sublinear time distributed algorithm for minimum-weight spanning trees, SIAM Journal on Computing 27 (1998) 302–316.

[8] S. Kutten, D. Peleg, Fast distributed construction of smallk-dominating sets and applications, Journal of Algorithms 28 (1998) 40–66.

[9] M. Elkin, A faster distributed protocol for constructing a minimum spanning tree, Journal of Computer and System Sciences 72 (2006) 1282–1308.

[10] Z. Lotker, B. Patt-Shamir, E. Pavlov, D. Peleg, Minimum-weight spanning tree construction in o (log log n) communication rounds, SIAM Journal on Computing 35 (2005) 120–131.