A DISTRIBUTED SYNCHRONOUS ALGORITHM FOR MINIMUM-WEIGHT SPANNING TREES

Md. Mohsin Ali¹, Mst. Shakila Khan Rumi²

¹Department of Computer Science, The University of Western Ontario, Canada
²Department of Computer Science and Engineering, Khulna University of Engineering and Technology, Bangladesh

Abstract: This paper presents a distributed synchronous algorithm for constructing the Minimum-Weight Spanning Tree (MST) in a connected undirected graph with distinct edge weights. Each node in the graph is considered as a processor having the initial knowledge of weights of the adjacent edges and each edge is considered as communication link. Each processor executes the same synchronous algorithm and exchanges the messages with the neighbours until the complete MST is formed. This algorithm constructs the fragments of MST and these fragments are then combined to form the complete MST. The total number of messages required for $N$ processors and $E$ communication links is $O(N^2 \log_2 N + E)$ and the time complexity is $O(N \log_2 N)$.

Keywords: Distributed, minimum-weight spanning tree, synchronous, communication complexity, time complexity.

I. INTRODUCTION

The problem of finding a distributed algorithm for an MST is a fundamental problem in the field of distributed network algorithms. Suppose $G(N, E)$ is a weighted connected undirected graph with $N$ nodes and $E$ edges where distinct weights are assigned to each edge and we want to find a spanning tree for which the accumulated weight of all its edges is minimized and denoted by MST. As for example, Figure 1 shows the graph of six nodes and ten edges denoted by the solid and dotted lines. The MST of the corresponding graph is the tree containing all the six nodes connected by the solid line edges. Moreover, we want to use a distributed synchronous algorithm to find that MST by replacing each node with a processor and considering each edge as a bidirectional and error-free communication channel/link. In this paper, graph and network, edge and link, and node or vertex and processor terms are used interchangeably. However, the processors can exchange messages among themselves over these communication channel. We know each processor initially knows the weights of each adjacent links and also able to determine the minimum among these one.

![Figure 1: A graph and the corresponding MST.](image_url)

Trees are an essential structure in various communication protocols and to deliver the information to every processor with low communication cost, it is advantageous to broadcast the information over an MST. Additionally, leader election is an important tool for breaking symmetry in a distributed network which makes the highly centralized protocols allow to run in a decentralized environment. This leader election problem can be reduced to the problem of finding an MST. Moreover, there are many other problems like counting the total number of processors in the network are closely related to the problem of finding an MST.
Figure 2: Construction of an MST for a given network.
An extensive research work has been done to develop efficient distributed algorithms for the MST problem. The study of the distributed algorithms for the MST construction was initiated by Gallager, Humblet, and Spira [1], which introduced a basic distributed technique for the problem and presented an algorithm with time complexity $O(N \log_2 N)$ and communicational complexity $O(N \log_2 N + E)$ on an $N$ processor network having total number of communication link $E$. The communicational complexity of this algorithm is optimal, but the time complexity is not. The time complexity was improved to $O(N \log_2 \log_2 N)$ by Chin and Ting [2]. Further this time complexity is improved to $O(N \log N)$ by Gafni [3], and then improved to existentially optimal running time of $O(N)$ by Awerbuch [4]. The $O(N)$ bound is essentially optimal because there exists graphs where no distributed MST algorithm can do better than $\Omega(N)$ time. However, the diameter $D(G)$ of a network $G$ for many networks significantly smaller than the number of processors $N$ and therefore it is a good criteria to design protocols whose running time is bounded in terms of $D(G)$ rather than $N$. One such algorithm considering this criteria was proposed by Garay, Kutten, and Peleg with the running time complexity of $O(D(G) + N^{0.61})$ [5] and further it was improved by Kutten and Peleg to $O(D(G) + \sqrt{(N \log_2 n)})$ [6].

An algorithm something like Gallager, Humblet, and Spira [1] but synchronous in fashion is proposed here with time complexity of $O(N \log_2 N)$ and communicational complexity of $O(N^2 \log_2 N + E)$.

The rest of the paper is organized as follows. Section 2 describes the high level description of the proposed algorithm and the correctness of the algorithm is discussed in section 3. The complexity analysis of this algorithm is provided in section 4 and section 5 concludes this paper.

II. PROPOSED ALGORITHM

Description: The proposed synchronous distributed MST algorithm constructs the MST parallelly from the weighted undirected connected graph/network having distinct edge weights and processor IDs.
Figure 4: Determination of MWOE of a fragment and informing this to remaining processors of the fragment.
The MWOE of any fragment except the single node fragment is determined and informed to all the processors of the fragment by the convergecast of $UpdateFL(F, L)$, $DetMWOE(MWOE)$, and $InformMWOE(MWOE)$ messages within that fragment before initiating the next round of joining the fragments. Figure 3 shows the process of determining the MWOE of a fragment with more than one processors. Here solid lines means these links are already included to the MST fragment and broken lines means these are available for inclusion in the MST fragment or rejected for inclusion. There are two fragments in this figure - one containing links (2, 1) and (1, 4) with LEADER=1 and level=3, and another containing link (6, 5) with LEADER=5 and level=2; which are created in the previous rounds and link (2, 4) is the rejected link selected in another round of the algorithm. In the next round, a new fragment containing links (2, 1), (1, 4), (4, 6), and (6, 5) is created with LEADER=1 and level=5 after joining the previously created two fragments. Now the MWOE of this new fragment needs to be selected first to start the next round. But this selection process in the new fragment starts as soon as the previously created two fragments initiate the process of joining together to create a new fragment.

In order to do this, the $UpdateFL(F, L)$ and $DetMWOE(MWOE)$ is generated by one of the processors sharing the MWOE having smaller ID (processor 4) just after combining the two fragments (with LEADER=1, level=3 and LEADER=5, level=2) by sending and receiving
The ConnectEdge(F, L) message over the common MWOE and send these two messages to all its adjacent fragment links. Here F is the LEADER identifier and L is the level number. The smaller processor ID of any fragment is denoted by F and the total number of processors in any Fragment is denoted by L. The purpose of the UpdateFL(F, L) message is to update F and L whereas that for DetMWOE(MWOE) is to update the MWOE of the new fragment. A processor updates its L and F as soon as it receives UpdateFL(F, L) message and forward this message to the adjacent fragment edges except the one from which it receives that. Moreover, it also initiates the message DetMWOE(MWOE) to send to all its adjacent fragment edges whenever it receives that. On the other hand, a processor updates its MWOE as soon as it receives DetMWOE(MWOE) message and forward this message to the adjacent fragment edges except the one from which it receives it. But the LEADER processor (processor 1) do not forward this message and waiting to receive the DetMWOE(MWOE) message from all the remaining processors (processor 2, 4, 6, and 5) of the fragment. If this LEADER processor receives this message from all its remaining processors of the corresponding fragment, then it makes sure that the determination of MWOE of that fragment is complete and informs this MWOE to all the processors of the corresponding fragment by sending InformMWOE(MWOE) message to all its adjacent fragment links of the new fragment. When a processor receives this message, then it updates it MWOE and forwards this message to the remaining adjacent new links of the new fragment except the one from which it receives that message. In this way, all the processors of the new fragment is informed about the MWOE of that fragment. As for example, Figure 4 shows the steps of determining the MWOE of a fragment and informing this to the remaining processors of the fragment, where the number inside the square box denotes the MWOE of the corresponding fragment and the arrow demotes the direction of the exchange of UpdateFL(F, L), DetMWOE(MWOE), and InformMWOE(MWOE) messages. Following this, the processors with MWOE of that fragment (processor 4 and 5) sends the ConnectEdge(F, L) message over this edge and (also the single processor fragments send this message over their MWOE) proceed to the next round. In this way, the MST is formed when there is only one fragment with all the processors of the network is available with no MWOE of that fragment and then terminates the algorithm.

Algorithm 3: Send_Messages()

```
1. Comment Sends different messages when these are available to a processor. if (MWOE ≠ null and OEList ≠ null and connect ≠ null)
2. if MWOE ∈ OEList then
3.   Send ConnectEdge(F, L) on MWOE;
4.   sendFlag ← MWOE;
5. if (update ≠ null) then
6.   Send UpdateFL(F, L) on connected edges except the edges on which it receives this message on previous round. If it does not receive this message (e.g. newly generated), then send to all connected edges;
7.   update ← null;
8. if (processor ≠ F and determine ≠ null) then
9.   Send DetMWOE(MWOE) on connected edges except the edges on which it receives this message on previous round. If it does not receive this message (e.g. newly generated), then send to all connected edges (null value of MWOE is allowed);
10. determine ← null;
11. counter ← 0;
12. if (inform ≠ null) then
13.   Send InformMWOE(MWOE) on connected edges except the edges on which it receives this message on previous round. If it does not receive this message (e.g. newly generated), then send to all connected edges (null value of MWOE is allowed);
14. inform ← null;
15. if MWOE = null then // MST complete
16.   return;
```

The pseudocode of the algorithms are shown in Algorithm 1-7.
It is mentioned previously that the proposed algorithm is something like the algorithm of Gallager, Humblet, and Spira [1], but synchronous in fashion. Moreover, there are many cases in which they are different. Their algorithm selects the minimum weight link of a fragment as a core of that fragment and the processors adjacent to this core only start the next round. A level is initially set to 0 and only increased by one when two fragments of the same levels are combined. If a fragment with level \( L < L_0 \) receives a connect message from another fragment with level \( L_0 \), then it simply waits to respond until \( L \) reaches a high enough level. On the other hand, our algorithm selects the processor with minimum ID in a fragment as a LEADER in that fragment and it takes the responsibility of informing the MWOE to the remaining processors of the corresponding fragment. The next round is started by the processor which has the smallest MWOE of the corresponding fragment. A level number in this algorithm is initially set to 1 and this is updated with the total number of processors in the fragment. In their algorithm, it is not necessary to exchange the connect message over the common link to combine the two fragments, but our algorithm it is necessary. The proposed algorithm takes the decision of acceptance or rejection of any link to the new fragment during the fragment combination stage of the algorithm; whereas it is determined during the selection of MWOE of their algorithm. Moreover, the number of different types of messages in our proposed algorithm is smaller than their algorithm.

**Algorithm 4: Receive_ConnectEdge\((F_1, L_1)\)**

Comment Process something when it receives \( \text{ConnectEdge}(F_1, L_1) \) message.

1. if \( \text{sendFlag} = e \) then
   
   2. if \( F \neq F_1 \) then /* Different fragment, no cycle */
   
   3. Accept this edge;
   
   4. \( F \leftarrow \text{Min}(F_1, F) \); /* Minimum of \( F_1 \) and \( F \) */
   
   5. \( L \leftarrow \text{Sum}(L_1, L) \); /* Summation of \( L_1 \) and \( L \) */
   
   6. \( \text{connect} \leftarrow \text{connect} \cup \{e\} \); /* Update connected list */
   
   7. \( \text{OEList} \leftarrow \text{OEList} \setminus \{e\} \); /* Update OEList list */
   
   8. \( \text{connect} \leftarrow \text{null} \);
   
   9. \( \text{sendFlag} \leftarrow \text{null} \);
   
10. if \( F < F_1 \) then /* Lower fragment processor among the common MWOE */

   11. \( \text{update} \leftarrow U \); /* U means \( \neq \text{null} \) */

   12. \( \text{determine} \leftarrow D \); /* D means \( \neq \text{null} \) */

13. else if \( F = F_1 \) then /* Same fragment, cycle */

   14. Reject this edge;

   15. \( \text{OEList} \leftarrow \text{OEList} \setminus \{e\} \); /* Update OEList list */

   16. \( \text{determine} \leftarrow D \); /* D means \( \neq \text{null} \) */

   17. \( \text{connect} \leftarrow \text{null} \);

   18. \( \text{sendFlag} \leftarrow \text{null} \);

else

19. Do nothing;

**Algorithm 5: Receive_UpdateFL\((F_1, L_1)\)**

Comment Process something when it receives \( \text{UpdateFL}(F_1, L_1) \) message.

1. \( F \leftarrow F_1 \);

2. \( L \leftarrow L_1 \);

3. \( \text{update} \leftarrow U \); /* U means \( \neq \text{null} \) */

4. \( \text{determine} \leftarrow D \); /* D means \( \neq \text{null} \) */
III. CORRECTNESS OF THE ALGORITHM

The correctness of the algorithm depends on the following four cases.

1. The algorithm does indeed find MWOE from the fragment,
2. The algorithm does not create any cycle,
3. The algorithm terminates properly, and
4. The algorithm finds the correct MST.

In order to prove the first case, there are two cases may happen - there is only one processor in each fragment and there are more than one processors in each fragment. It is assumed that the processor is able to find out its own MWOE for the single processor fragment and the algorithm exchanges the UpdateFL(F, L), DetMWOE(MWOE), and InformMWOE(MWOE) messages following the exchange of ConnectEdge(F, L) message described in algorithm to find the MWOE of the fragment under the assumption of distinct edge weights.

In order to prove the claim that there is no cycle creation in the algorithm, we have to observe two cases. The first one is that the algorithm does not include any links in the MST such that it creates the cycle. And the second one is that the exchange of ConnectEdge(F, L), UpdateFL(F, L), DetMWOE(MWOE), and InformMWOE(MWOE) messages occur for a finite number of times. The cycle creation avoidance for the first case is checked by the value of F during the exchange of ConnectEdge(F, L) message over the common MWOE. If the value of F of the processor receiving this message and the F of the message received are same, then the processors exchanging this message are in the same fragment and inclusion of this link in the MST creates a cycle. The algorithm properly handles this case and the result is no cycle. On the other hand, the corresponding ConnectEdge(F, L), UpdateFL(F, L), DetMWOE(MWOE), and InformMWOE(MWOE) messages are initiated and send to all adjacent links in the fragment by the corresponding single processor (processor with smaller ID exchanging ConnectEdge(F, L) message) in that fragment, but forwarded to only the adjacent links in the fragment except the one from which it receives that. As a result, the exchange of these messages will stop after a finite number of rounds and there is no cycle.

Moreover, the exchanges of InformMWOE(MWOE) message to the other remaining processors in the fragment with the MWOE parameter as null by the LEADER processor indicates that there is no MWOE of the fragment. This situation will happen only once in the network when there is only one fragment in the network indicating the completion of the
MST generation and the corresponding processor terminates from the algorithm upon receiving this message. As a result, the algorithm properly terminates from the system.

However, it is showed in [1] that it is sufficient to verify that the algorithm does indeed find MWOE from the fragment and that the waiting does not lead to deadlock in order to prove that the algorithm correctly finds the MST of the network provided. So, after proving the above three cases, we can say that the algorithm correctly finds the MST of the network provided. So, after proving the above three cases, we can say that the algorithm correctly finds the MST of the network provided. 

So, the proposed distributed synchronous algorithm correctly finds the MST.

IV. ANALYSIS OF ALGORITHM

1. Communicational Complexity

In order to determine the communicational complexity of this algorithm, it is considered to determine the upper bound on the total number of messages exchanged during the execution of the algorithm. The message size depends on some parameters that it holds. The complex message contains one $F$, one $L$ (total number of processors in the fragment), and few bits to indicate the message type; or one MWOE and few bits to indicate the message type.

Each link can be connected or rejected at most one and it exchanges at most two messages ($ConnectEdge(F, L)$). So for the total number of links $E$ of the network, it sends at most $2E$ messages which is $O(E)$.

Moreover, at each level (with the progression of the MST), at most one processor (processor with smallest ID having common MWOE) sends at most two $UpdateFL(F, L)$ messages and the remaining processors of the same level of the fragment send at most one message ($UpdateFL(F, L)$) with a total of $O(N \log_2 N)$ messages for all $\log_2 N$ levels. At the last level, one processor sends at most $N$ $DetMWOE(MWOE)$ messages, one sends at most $N - 1$ of that messages, . . . , one sends at most 1 of that messages, and as a total of at most $N(N+1)/2$ messages are sent for that last level to the LEADER. Then for all the levels ($\log_2 N$), it sends at most $O(N^2 \log_2 N)$ $DetMWOE(MWOE)$ messages. Upon completion of determination of the local MWOE of the processor, one processor (LEADER) sends at most two and the remaining send one $InformMWOE(MWOE)$ messages to a level to synchronize the MWOE of the corresponding fragment. So it sends also a total of $O(N \log_2 N)$ messages for all $\log_2 N$ levels. Additionally, except the $2E$ messages, some processors are continued to send the $ConnectEdge(F, L)$ messages until they receive the same type of message at the same time on the common MWOE to include this edge on the MST. But the inclusion of these messages are unnecessary as it is considered that $N$ processors send the $UpdateFL(F, L)$, $DetMWOE(MWOE)$, and $InformMWOE(MWOE)$ messages and these additional $ConnectEdge(F, L)$ messages are already considered in these messages.

As a result, the total communicational complexity of this algorithm is $O(N^2 \log_2 N + E)$.

2. Time Complexity

In worst case, it takes at most $3N$ rounds to exchange the $ConnectEdge(F, L)$, $UpdateFL(F, L)$, $DetMWOE(MWOE)$, and $InformMWOE(MWOE)$ messages in order to update $F$ and $L$, determine MWOE, and inform this MWOE to all the processors in a fragment. This worst case will happen when a processor with ID equals to $F$ (LEADER identifier) and the processor with local MWOE equals to fragment MWOE are in maximum distance after the combination of the previous fragments and one of the links of a processor with the same ID and $F$ are in the common MWOE of the previous fragments. Since, there are a total of $\log_2 N$ levels and so it takes at most $3N \log_2 N$ levels which is $O(N \log_2 N)$.

V. CONCLUSION

In this paper we reviewed some distributed algorithms for constructing MST with their communicational complexity as well as time complexity. Following this, we propose a synchronous distributed algorithm that constructs the fragments of the MST by exchanging some synchronous messages among the processors. These fragments then exchanging these messages again to construct the larger fragment of MST and eventually constructs the complete MST. Moreover, the communicational complexity and the time complexity is also measured following the correctness of this algorithm.

The communicational complexity of this algorithm is in the quadratic order of the total
number of processors in the network. The main part of the algorithm responsible for this worst complexity is the determination of MWOE of a fragment. This determination of MWOE is centralized to the LEADER to determine the MWOE as well as informing this to the remaining processors of the corresponding fragment. The communicational complexity of this algorithm can be further improved by resolving this centralized behaviour and considering the exchange of total messages as minimum as possible.

REFERENCES


