Eighth International Conference on Distributed Computing Systems, June, 1988, San Jose, pp. 334-340.

A NEW NODE-JOIN-TREE DISTRIBUTED ALGORITHM FOR MINIMUM WEIGHT SPANNING TREES

Yao-Nan Lien

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210-1277
(614) 292-5236, lien@cis.ohio-state.edu, cbosgd!osu-cis!tut!lien

ABSTRACT

This paper develops a new distributed algorithm that uses Node-join-tree approach for the minimum spanning tree problem in a communication network. The algorithm needs at most $(2e + n(n-1)/4)$ messages in $O(n^2)$ time on a general random graph. In the best case, it needs only $2e$ messages in $O(logn)$ time. The parameters e and n are the number of edges and the number of nodes. Most existing algorithms use a Tree-join-tree approach and require at least $(2e + 2\ n\ \log\ n)$ messages and at most $(2e + 5\ n\ \log\ n)$ messages. Although the worst case performance of proposed algorithm is not better than Tree-join-tree algorithms under some extreme cases, it provides better performance in most cases as shown in our simulation study. The proposed algorithm is initialized from a single node such that there is no need to wake up all nodes at the beginning. Further, the algorithm is less complicated than other algorithms such that a reliable implementation is easier to achieve. Our results can be used to improve the algorithms for many other problems in distributed computing, such as Leader-election, node-counting, Deadlock-resolution, and Message-broadcasting.

1. INTRODUCTION

Given a graph with a set of nodes and a set of weighted edges, the *Minimum weight Spanning Tree* (MST) is the set of edges that connect all nodes into a tree with the minimum of total weight.[3] The *Distributed Minimum weight Spanning Tree problem* (DMST) is to find a minimum spanning tree in a communication network with minimum number of messages. Each node in the network knows only its adjoining edges initially, executes the same local algorithm, and knows which of these edges are on the MST at the end.[7] The nodes in the graph are initially asleep. In our algorithm, one of them wakes up to initiate the algorithm. There is no need to wake up all nodes to initiate the algorithm. Every other node is waken up by the first message sent from its neighboring nodes. After waken up, a node executes a local algorithm which consists of sending messages over adjoining links, waiting for incoming messages, and processing. Messages can be transmitted independently in both directions on each edge and arrive after an unpredictable but finite delay, without error and in sequence.[7]

The DMST is one of the most important problems in distributed computing systems. The Leader-election, Deadlock-resolution, Node-counting, and Message-broadcasting problems are the examples that can be modeled as a DMST problem. A MST provides the most communication efficient solution to these problems. Other examples can be found elsewhere.[2]

Current best approach to solve the problem is to iteratively join the fragments of an MST into a complete one, called the *Tree-join-tree* approach for convenient. In this approach, an MST grows in a way like an MST grows in Prim's and Dijkstra's sequential algorithms.[5,10] The worst case complexity of this approach is approaching its theoretical limit.[8,1] This paper proposes a different approach, called the *Node-join-tree* approach, to improve the average case performance. In the new approach, a single MST fragment grows from a single node to a complete tree. In this approach, an MST grows in a way like an MST grows in the Kruskal's sequential algorithm.[5] In the best case, the algorithm needs only $2e$ messages. Under some conditions, the worst case message complexity is comparative with the Tree-join-tree approach.

This paper is organized as follows. Some definition and properties of MST are described in the rest of this section. Previous work is given in Section 2. Section 3 details the new proposed algorithm with a brief description of the communication network model. Its correctness and performance are discussed in Section 4. The conclusion and future work are given in Section 5.

Definitions and Properties of MST

A connected subtree of an MST is called a *fragment*, which is also an MST with respect to the subtree. An edge is an *outgoing edge* of a fragment if one adjacent node is in the fragment and the other is not. An edge in a particular MST is referred to as an *M-edge* of the MST. Given any MST, a consistent precedence relationship among all nodes with respect to any node, called the *root*, is uniquely defined. The defined precedence relationship is referred to as the *rooted-MST* with respect to the tree and the selected root. For a node in a rooted-MST, the edges (not necessary to be M-edges) connected to its preceding nodes (resp. succeeding nodes) are referred to as the *upward edges* (resp. *downward edges*). An upward M-edge (resp. downward M-edge) of a node is called an *up-hook* (resp. *down-hook*) of the node. Each non-root node has a unique uphook. Other edges are called *noward edges*. With respect to a particular rooted-MST, the *up-end* of an M-edge is the end connected to the parent node and the *down-end* is the end connected to the child node. An example of rooted-MST with a 10 node netowrk is shown in Figure 1. Node a is the root node of the rooted-MST shown in the graph. With respect to node f and the rooted-MST, a and e are the preceding nodes; h, i, and j are the succeeding nodes; (f, a) and (f, e) are upward edges; (f, h), (f, i) and (f, j) are downward edges; (f, c), (f, d) and (f, g) are noward edges; (f, e) is the up-hook; and (f, h) and (f, i) are down-hooks.

Some important properties of an MST in a graph are summarized as follows.

P1. Each node defines a unique rooted-MST rooted by the node.

P2. With respect to a particular rooted-MST, each non-root node has exactly one parent node, and may have more than one child node.

The Eighth International Conference on Distributed Computing Systems, San Jose, June 1988

ing process iteratively until the end of the iteration when there is no node wants to hook to M. A complete MST is formed if all nodes are included in M. Otherwise, the minimum outgoing edge of M is determined, called the *core* of the iteration, and M hooks itself to the node at the other end of the core. The newly selected node is designated as the new root node to initiate a new iteration. This procedure is repeated until a complete MST is found.

There are two methods to identify the minimum edge for a fragment. In the first method, called *short-message method*, the messages in the communication network have a very limited capacity such that only minimal information can be carried in the messages. All nodes in a fragment have to cooperate to each other to identify the minimum weight outgoing edge. In this method, the fragment itself is used as a comparing tree. The root node sends messages to all terminal nodes to initiate the process. Each terminal node sends its identification and the weights of its minimum outgoing edge to its parent node. Each non-terminal node then selects the one with the minimum weight among its descendents and reports to its parent node. The minimum edge will be selected by the root node of the fragment finally. The root node notifies the selected node to terminate the process. Since all nodes in the fragment are involved in the process, at least $2m$ messages are required to identify the minimum weight outgoing edge, where m is the number of nodes in the fragment. This method is employed in most Tree-join-tree algorithms and needs up to $5n$ messages in each iteration. The details of this method can be found in Gallager's paper.[7]

In the second method, called *long-message method*, each message has a larger capacity to carry more information for the minimum outgoing edge identification. Although this assumption is not generally true in all distributed computing systems, but it is valid in many practical communication networks, in which a long message may have the same or close overhead as a short message due to the inevitable networking overhead. As a mater of fact, the communication overhead is usually counted in packets which may have a capacity up to 1000 bytes in many networks. In this long-message method, the identification of all nodes adjacent to M and the weights of the edges connecting these nodes to M are sent to the root node in the replying messages. The minimum edge and the new root node are identified by the current root node itself. Then, the entire information is forwarded to the new root node. There is no need to collect the information again in latter iterations. The information can be split into several messages if the information to be carried exceeds the capacity of a message. Although this will increase the message count slightly, this method remains a practical alternative in many systems.

Only the long-message method will be detailed in this paper since the short-message method has been extensively studied.

3.3. Algorithm DMST-NJT

For convenient, the following colors (states) are used to mark the edges and nodes during the tree growing process.

Red: The edges not in M but connecting nodes in M are marked red. In other words, including any red edge into M will creat a cycle in M.

Green: The first root node and all non-root nodes of M are marked green. All but core edges of M are marked green.

Blue: All root nodes except the first one are marked blue. All core edges are marked blue.

Yellow: The node that is adjacent to M and has rejected at least one 'Follow-me' message is marked yellow. The edge connecting a yellow node to M is marked yellow.

yellow nodes may change to either green or blue and yellow edges may change to either blue or red.

White: All other edges and nodes are marked white.

Note that there is no red nodes since every node must be in M eventually. Only red, blue, and green are permanent. Comparing to the state definitions in Gallager's paper, green and blue edges are the edges marked as Branch; white and yellow edges are the edges marked as Basic; and red edges are the edges marked as Reject.

The algorithm to be executed in each node is detailed as follows. Step 1 and Step 5 are only executed at the root node of each iteration. While Step 2 to Step 4 are executed at other nodes.

0. Initially, all edges and nodes are marked white. All nodes are in the state 'Asleep'. An arbitrary node waken up as the first root node and mark itself green. The node then executes Step 1 to initiate the process.

1. The new root node sends a 'Follow-me' message to each of its neighboring nodes except those already in M and then enters the state 'Wait-for-terminate'.

2. Upon receiving a 'Follow-me' message from a particular edge, a node is waken up and enter the state 'Wait-for-draft' if it is in the state 'Asleep'. Once it is waken up or is already waken up, it executes one of the following steps:

   (a) If it is already in M and is in the state 'Wait-for-reply', it marked the edge red.

   (b) If it is already in M and is in the state 'Wait-for-draft', it replies a 'Red' message and marked the edge red.

   (c) If it is in the state 'Wait-for-draft' and its minimum edge is not emanating to the drafting node, it replies a 'Yellow' message with the identity of the node itself. The edge and the node itself are marked yellow.

   To reduce overhead, a yellow node can reply with a 'Red' messages and mark the edge red if the weight of the edge is not smaller than that of yellow edges it has marked. Yellow nodes are the neighboring nodes of M. They are waiting for the drafts from other nodes or for a chance to be a new root.

   (d) Otherwise, its minimum edge is emanating to the drafting node. The node decides to hook itself to M as a new terminal node of M. All yellow edges adjacent to the node are changed to red. Then, it initiates a new drafting process by sending a 'Follow-me' message to all neighboring white edges. Finally, it enters the state 'Wait-for-reply' and waits for the replies.

3. Upon receiving a reply from an edge, a node in the state 'Wait-for-reply' will mark the edge according to the color of the replying message. It will either keep wiating or execute Step 4 if all replies have been received.

4. Upon receiving all replies, a node marks itself and the edge emanating to the drafting node as green and reports a 'Green' message to the drafting node. Then it enters the state 'Idle'. As a matter of fact, the edge emanating to the drafting node is its up-hook with respect to the current root node. The content of a

number in the 'Follow-me' and 'Red' messages such that each node in M knows at which iterations its neighbors and itself are included. The yellow nodes in the 'Root-migration' messages are also associated with iteration numbers. In this way, a node can forward the message to the neighboring node that is in the iteration closest to that of the new root node. The number of hops required in root-migrations can be reduced.

3.5. Remark

There is no need to wake up all nodes at the beginning. Further, the algorithm is simple such that it is very easy to implement in a distributed processing system. In general, the implementation of a distributed algorithm is much more difficult than that of a sequential algorithm. It is more difficult to test, debug, and verify due to the asynchronous nature of the environment. The lack of a global consistent clock makes the problem even more critical. Therefore, the simplicity should be considered as a very important parameter in the evaluation of distributed algorithms.

4. CORRECTNESS AND PERFORMANCE

4.1. Correctness

Theorem 1: *Algorithm DMST-NJT generates a minimum weight spanning tree for any graph.*

Proof: Since each node joins to the MST (M) through a unique edge (either green or blue) in the tree growing process, a consistent precedence relationship among these nodes can be defined so that it must not contain any cycle. Next, we prove that it is a spanning tree. We assume that there is such a node not being included in M after the process is terminated. This node must have rejected all 'Follow-me' messages sending by its neighboring nodes and have not been selected as a root node. However, its neighboring nodes must have reported the information to the roots in various iterations. In this case, it must have been selected as a root node. This is a contradiction, therefore.

Finally, we prove that M is a minimum weight spanning tree. In each iteration of the algorithm, the active fragment itself is a minimum weight spanning tree (a fragment), because the up-hook (with respect to the current root node in the active fragment) of each green node is the minimum weight outgoing edge of the node. It can be proved using property P4 that, in any iteration, the core connects the idle fragment and the active fragment into a bigger fragment because the core is the minimum weight outgoing edge of the idle fragment. Therefore, it can be proved by induction that the spanning tree obtained in the last iteration is a minimum weight spanning tree.

□

4.2. Communication Complexity

The performance of a distributed algorithm is evaluated by the *communication complexity* and the *time complexity*. Because that the communication complexity, which is measured by the number of messages, dominates the computation complexity in many applications, the computation complexity is usually ignored unless it is significantly high.

The messages needed in this algorithm can be divided into two sets, the first set for node drafting and the second set for root-migrations. The first set of messages is exactly $2e$ regardless of the topology and at which node the process is initiated. It is shown in our simulation study that the number of root-migrations in most of graphs are only minimal when n is not large[9] such that the total number of messages is close to $O(e)$. However, it is very difficult to find an analytical estimation for the average number of messages required by root-migrations. The analysis of the worst case performance will be presented in the rest of this section.

The first set of messages is counted as follows. Exactly one 'Follow-me' message and one reply message (either 'Red', 'Green', or 'Yellow') travel through each edge. These two types of messages travel only one hop and there are $2e$ such messages in total. As a matter of fact, two adjacent nodes may send 'Follow-me' messages to each other at the same time. In this case, no 'Red' message will be replied so that the message count will not be changed.

Thus, the algorithm needs $2e$ messages in the best case when there is no any root-migration. In Tree-join-tree algorithms, at least $2e + 2n\log n$ messages are needed. It is shown in our simulation studies that the number of messages required by our algorithm is very close $2e$ in most cases since the number of root-migrations is only a minimal.

Next, we estimate the total number of root-migrations in the worst case. The worst case happens when all nodes connected to a single line ($d=n-1$), every other edge is blue, and each 'Root-migration' message has to travel from one end to the other end of M as shown in Figure 3. The worst case will not happen when every edge is blue since each 'Root-migration' message will travel only one hop in this case as proved in Lemma 2.1. A root-migrations is called a *back-to-back root-migration* if it happens immediately after the previous root-migration. That is, no green node is generated between the two consecutive root-migrations. It is shown in Lemma 2.1 that a back-to-back root-migration can only happen when two nodes are adjacent to each other.

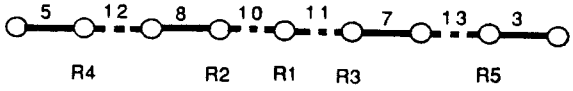


Figure 3. A worst case problem instance.
(R_i is the root of the ith iteration.)

Lemma 2.1: *A back-to-back root-migration may only happen between two adjacent nodes.*

Proof: Suppose that a node n_b is selected as the new root node at the end of an iteration and e_b is the core connecting n_b to the old fragment (the idle fragment) as shown in Figure 4. The weight of e_b must be the minimum of all current yellow edges. Also, there exists at least one white edge e_b' incident to n_b whose weight is smaller than e_b. Otherwise, n_b must be already drafted into M by the node at the other end of e_b. If a back-to-back migration happens, e_b' must be the minimum weight outgoing edge of the new fragment.

□

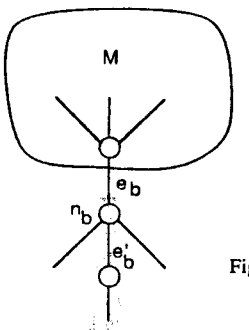


Figure 4. Proof of Lemma 2.1.

From simulation, we observed that the number of root-migrations has a complexity of $O(n)$. However, the number of nodes in each iteration is less than n such that U is still small comparing to $n \log n$ in the Tree-join-tree algorithms. Also, there is a weak connection between the degree of graphs and the message complexity since the number of root-migrations depends on the degree of the graph. For a constnat n as the degree d increases, then so does the probability of drafting nodes. Since a root-migration has to take place only if when no more nodes can be drafted. Note that U shown in Table 1 is almost independent of d. However, U is only an upper bound for the message measure in root-migrations. The real number of messages depends on the depth of M, rather than the number of node in M. When the degree increases, the depth of M decreases at a fixed number of node in M.

It is clear that the upper bound of average messages is smaller than that of other algorithms when n is moderate. Since U in Table 1 is large when n is large, the comparison is difficult. However, we would like to suggest to add the iteration numbers in the long-message method such that the number of hops needed by a root-migration can be reduced to $\log |M|$ level. In this case, the proposed algorithm will be better than others even if n is large. Another possibility is to use a hybrid method when n is large that uses the proposed algorithm to build many smaller fragments and then uses Tree-join-tree algorithms to combine these fragments into a complete MST.

Table 1 Simulation Results of Average Performance.

n	d	RM	U	$2e+U$	$2e+2U$	$2e+5U$	$2e+2n\log n$	$2e+5n\log n$
10	3	3.03	22.83	52.83	75.66	144.15	110	230
10	4	3.05	27.38	67.38	94.76	176.90	120	240
10	6	4.0	73.79	133.79	207.58	428.95	140	260
10	8	2.86	47.95	127.95	175.90	319.75	160	280
20	3	7.0	70.9	130.9	201.8	414.5	260	560
20	4	6.7	79.12	159.12	238.24	475.60	280	580
20	6	6.37	98.86	218.86	317.72	614.30	320	620
20	8	6.4	121.8	281.8	403.6	769.0	360	660
50	4	18.88	295.04	495.04	790.08	1675.20	800	1700
50	6	16.36	263.84	563.84	827.68	1619.20	900	1800
50	8	18.12	296.24	696.24	992.48	1881.20	1000	1900
50	10	15.24	268.64	768.64	1037.28	1843.20	1100	2000
100	4	37.2	1220.64	1620.64	2841.28	6503.20	1800	3900
100	6	36.2	1223.68	1823.68	3047.36	6718.40	2000	4100
100	8	33.72	1179.96	1979.96	3159.92	6699.80	2200	4300
100	10	34.68	1159.92	2159.92	3319.84	6799.60	2400	4500

5. CONCLUSION AND FUTURE WORK

A distributed algorithm based on a new Node-join-tree approach for the minimum spanning tree problem in a communication network is proposed in this paper. The new algorithm makes a good utilization of the properties of the minimum weight spanning tree to improve the existing algorithms in the average cases. The new algorithm provides a comparative worst case and better best case message complexity than the existing algorithms that are based on the Tree-join-tree approach. The algorithm needs at most $(2e + n(n-1)/4)$ messages in $O(n^2)$ time. IN many cases, it needs only $O(e)$ messages in $O(\log n)$ time when n is not very large. The proposed algorithm is initialized from a single node such that there is no need to wake up all nodes at the beginning. Furthermore, the algorithm is less complicated than other algorithms such that a reliable implementation is easier to achieve. Simulation results show that the average performance of the proposed algorithm is generally better than other algorithms on regular graphs.

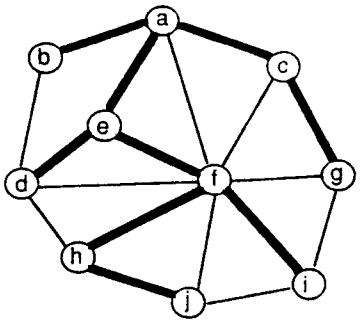
In the future, the methods to reduce the number of hops required by root-migrations such as the one proposed in this paper will be studied. Also, a hybrid method that combines the Tree-join-tree and Node-join-tree approaches will be studied to improve the performance when the long-message method is not applicable.

Acknowledgement

The author wishes to thank Professor Steve Lai and Professor Chin-Lung Lei of the Ohio State University, and Professor Xin He of SUNY Buffalo for their precious comments.

References

1. Awerbuch, Baruch, O. Goldreich, and R. Vanish, "On Message Complexity of Broadcast: a Basic Lower Bound," *unpublished manuscript*, Jan. 1987.
2. Awerbuch, Baruch , "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and related problems," *Proc. of the 9th Annual ACM Symposium on Theory of Computing*, pp. 230-240, New York City, May, 1987.
3. Bondy, J. A. and U. S. R. Murty, in *Graph Theory with Applications*, North-Holland Publications, New York, 1976.
4. Chin, F. and H. F. Ting, "An Almost Linear Time and O(nlogn+e) Messages Distributed Algorithm for Minimum-Weight Spanning Trees," *Proc. of 1985 FOCS Conference*, pp. 257-266, Portland, Oregan, October 1985.
5. Dijkstra, E., "Two Problems in Connection With Graphs," *Numer. Math.*, vol. 1, pp. 269-271, 1959.
6. Gafni, E., "Improvements in Time Complexity of Two Message-optimal Algorithms," *Proc. of 1985 PODC Conference*, Minacki, Ontario, August 1985.
7. Gallager, R. G., P. A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Trans. on Program. Lang. & Systems*, vol. 5, pp. 66-77, Jan. 1983.
8. Korach, E., S. Moran, and S. Zaks, "Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of Processors," *3rd ACM Symp. on Principles of Dist. Comp.*, pp. 199-207, Aug. 1984.
9. Lien, Yao-Nan, "The Perforamance Evaluation of a New Node-join-tree Distributed Algorithm for Minimum Weight Spanning Trees," Technical Report, Dept. of Comp. and Info. Sci., The Ohio State Univ., 1988.
10. Prim, R. C., "Shortest Connection Networks and Some Generalizations," *Bell System Tech. J.*, vol. 36, pp. 1389-1401, 1957.



Root Node	a
Current Node	f
Preceding Nodes	a, e
Succeeding Nodes	h, i, j
Upward Edges	(f,a) , (f,e)
Downward Edges	(f,h), (f,i), (f,j)
Noward Edges	(f,c), (f,d), (f,g)
Uphook	(f,e)
Downhook	(f,h), (f,i)

Figure 1. An example of a root_MST on a graph.

P3. With respect to any non-root node in a particular rooted-MST, the weight of the up-hook is the minimum among all upward and noward edges.

P4. Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.

Property P3 can be easily proved as follows. Assuming that there exists an upward-edge or noward-edge with a weight smaller than that of the up-hook of a particular node, we can replace the up-hook with this edge to obtain another spanning tree with a smaller total weight. This is a contradiction.

Other properties can be found in Gallager's paper.[7] These properties are useful in designing an efficient algorithm. The proposed Node-join-tree approach makes a good utilization of Property P3 to offer better average performance.

2. PREVIOUS WORK

Current most efficient algorithms are based on Tree-join-tree approach and are proposed by Gallager, Humblet, and Spira.[7] The algorithm is then improved by Awerbuch,[2] Gafni,[6] and Chin and Ting.[4] The communication complexity of these algorithms is $O(e+n \log n)$ in both average and worst cases, where e is the number of edges and n is the number of nodes. More specifically, they need about $(2e+5n \log n)$ messages. Awerbuch improves the algorithm from nonlinear time complexity to $O(n)$ time complexity by counting the nodes at the beginning.[2] The worst case performance of these algorithms is approaching the theoretical upper bound.[1,2] However, the average case performance is yet to be improved. The basic scheme of Tree-join-tree approach is described as follows.

> The algorithm maintains a set of subtrees, each being a fragment of the MST to be constructed. Initially, every fragment consists of a single node. These fragments join to each other until a single fragment is found. Every fragment finds its best edge, the minimum weight outgoing edge, to *join* (or hook) itself to another fragment. The algorithm terminates when a complete MST is constructed.

One deficiency of this approach is the need to wake up all nodes either synchronously or asynchronously to initiate the algorithm. This initial wake-up process needs extra messages and may be impractical in real distributed computing systems. Another deficiency is the need to identify the minimum weight outgoing edge for each fragment generated in the intermediate stages. Each identification process may need n messages in the worst case. This process makes the distributed algorithm more complicated and may hurt the resiliency of the system.

This minimum weight outgoing edge identification induces a significant communication overhead explained as follows. The process essentially is to identify the edge that satisfies the conditions of Property P4 such that the fragment can be expanded. This process must be performed every time a new fragment is formed. As a matter of fact, the communication complexity of Tree-join-tree approach is dominated by these minimum edge identification processes.

After examining property P3 carefully, it is clear that it is easier for the down-end node of an edge to decide whether or not to include the edge into the MST. This decision is made at the up-end node in the Tree-join-tree approach and is made at the down-end node in the Node-join-tree approach. Nodes, instead of trees, hook themselves to a single MST fragment (M) until an MST is completely constructed. In the algorithm execution, the terminal nodes of M try to draft their neighboring nodes into M. Each neighboring node decides to hook itself to one of the drafting nodes based on its own local information. This decision is fairly easy to make as to be described in Section 3. This basic difference makes the Node-join-tree approach more efficient than the Tree-join-tree approach in the majority of cases. There is no need to wake up all nodes at the beginning. Furthermore, the algorithm is easier to implement in a distributed system.

3. PROPOSED ALGORITHM

In this section, we detail the proposed algorithm. The communication network model and the overview of the algorithm are described briefly in Section 3.1 and Section 3.2.

3.1. Communication Network Model

The concerned communication network is the standard model of static asynchronous networks.[7] This is a point-to-point message (or packet) switching communication network, represented by an undirected graph $G(V, E)$ where the set of nodes $V(|V|=n)$ represents the switching elements of the network and the set of edges $E(|E|=e)$ represents bidirectional communication links operating between neighboring nodes. Each edge is associated with a weight, which is system dependent. Each node knows only the edges emanated from itself. Nodes are labeled, but edges may not. Nodes communicate with each other by exchanging messages which are forwarded from node to node. It is assumed that a message takes one unit time to travel from the end of an edge to the other end (one hop). Due to the networking overhead, this assumption remains valid regardless the message length up to a certain limit.

3.2. Overview of the Algorithm

Starting from any node, an MST fragment (M) grows from a single node to a complete MST iteratively by *drafting* nodes into M. In each iteration, each terminal node of M tries to draft more nodes into M by sending a 'Follow-me' message to each of its neighboring nodes except its preceding node. Each neighboring node decides whether or not to hook itself to M as a new terminal node based on its own local information. The new terminal nodes continue the draft-

'Green' message depends on which minimum weight identification method is used. If the short-message method is used, it is only a simple message. Otherwise, it contains more information as to be described in Section 3.4.

5. Upon receiving all replies from all descending nodes, the root node will terminate the process if all nodes are already in M, that can be determined when there is no yellow node is reported. Otherwise, a new root node must be selected to continue the process. The yellow edge with the minimum weight is chosen as the '*core*' of next iteration and is marked blue. The node on the far end of the core is then designated as the new root node and is mark blue. Finally, a 'Root-migration' message is sent to the new root node and the new root node continues the process by executing Step 1. The root change is referred to as a *root-migration*. For convenient, the old M is referred to as the *idle fragment*. The set of nodes and edges that join to M after the latest root-migration is referred to as the *active fragment*. The minimum edge identification and root-migration are detailed in Section 3.4.

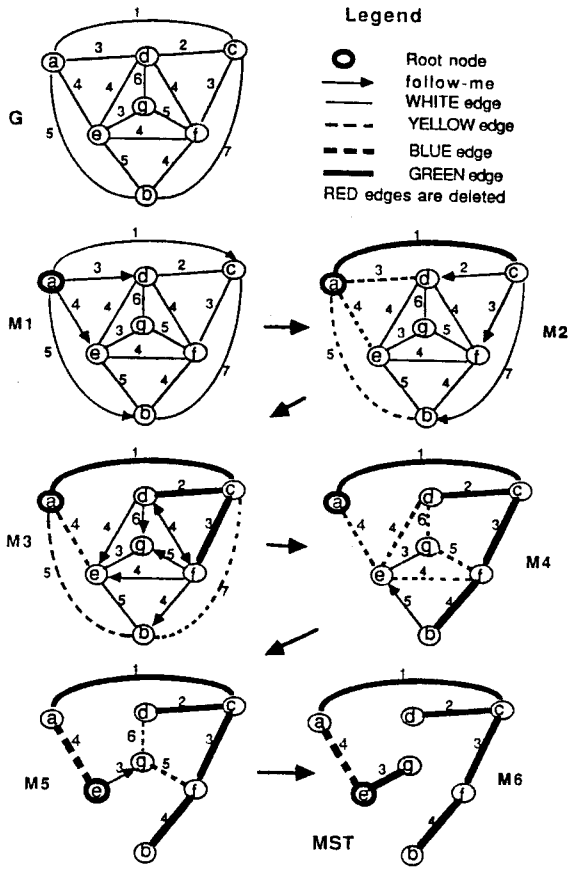


Figure 2. An example of MST growing process in algorithm DMST-NJT.

An example of the algorithm executed on a 7 node graph is shown in Figure 2. As we can see from the algorithm, the decision to hook a node to M is totally based on the node itself. There is no need to invoke a distributed process to make such decisions unless a root-migration is needed. Since the weights of edges need not to be distinct, more than one minimum weight edge may emanate from a node. In this approach, the node may hook itself to any of such minimum weight edges. However, hooking to the first node that sends the 'Follow-me' message along one of such edges will reduce unnecessary root-migrations.

3.4. Root-migration

As mentioned in Section 2.2, the short-message method will not be discussed here since it is a well studied method.[7] This section will concentrate on the long-message method. In this method, the yellow node information is contained in the 'Green' message and is propagated to the root node such that the new core and the new root node can be selected in the current root node itself. The content of the messages, the refinement of messages, and the routing algorithm of 'Root-migration' messages are discussed in this section.

The 'Green' message reported by a green node consists of the identities of all yellow nodes and green nodes in the subtree rooted by the node. Each yellow node is associated with the weight of the yellow edge connected to the node. There is a possibility that a yellow node sends inconsistent replies to different drafting nodes. Thus, each green node has to refine the information to eliminate the redundancy and inconsistency before it forwards the information to its parent node. In the refinement, if a node is ever marked green in any message reported by one of the descendent, the node will be marked as green. If a node is marked as yellow in all messages, the node will be marked yellow and only the reply with the minimum weight will be kept. Thus, each yellow node is associated with only one weight after the refinement.

It is essential to report the identities of green nodes to the preceding nodes. A node would be considered a yellow node if no green message is reported. In the root node, the identities of green nodes will be removed after the yellow node information is refined.

Because the number of green nodes will be much more than the number of yellow nodes, the identities of green nodes in 'Green' messages may present a great burden on the communication system. It can be reduced by a simple mechanism as follows. When a green node reports its identity to its parent node, it will report the number of 'Yellow' messages it has sent. This number will be reduced by one each time a yellow node information is removed. The green node information can be removed if this number is reduced to zero.

Since the global topology of the network is not known to any node, it is important to embed a distributed routing mechanism in the algorithm to forward the 'Root-migration' messages to their destinations. The routing is done similar to the most of routing algorithms in packet switching communication network. The entry associate to a particular yellow node in a routing table is the edge that the particular yellow node can be reached. The routing table can be set up from the 'Green' and 'Yellow' messages received by a node. When a green node receives a 'Green' or a 'Yellow' message through a particular edge, the yellow nodes included in the message can be reached from this edge. The routing table will be updated accordingly. If a 'Root-migration' message is to be forwarded to one of these yellow nodes, the corresponding edge will be chosen to forward the message. Otherwise, the message will be forwarded to its preceding node. Note that the precedent relationship mentioned here is referred to the original relationship when it is defined at the first time. In this way, there is no need to update the routing table. This routing mechanism may need up to n hops to forward a 'Root-migration' message in the worst case, especially, when the message is forwarded from the root node of a very late iteration to a yellow node of a very early iteration. One way to improve the routing mechanism is to incorporate the iteration

Each back-to-back 'Root-migration' message travels only one hop. From Lemma 2.1, it is easy to derive that the maximum number of multi-hop root-migrations in a graph is at most $(\lceil n/2\rceil-1)$.

Lemma 2.2: *For any given graph, there are at most* $(\lceil n/2\rceil-1)$ *multi-hop 'Root-migration' messages in Algorithm DMST-NJT.*

Proof: From Lemma 2.1, we know that a multi-hop root-migration must not be a back-to-back root-migration. Thus, there is at least one node marked green before a multi-hop root-migration. Since each blue node must be adjacent to at least one white node before it is included into M, the last node to be included into M must not be a blue node. If there were more than $(\lceil n/2\rceil-1)$ multi-hop root-migration generated, the total number of nodes would be greater than n including the first root node. This is a contradiction.

□

Using the same argument, it can be shown that the single-hop root migration will not happen when there are $(\lceil n/2\rceil-1)$ multi-hop root-migrations.

Finally, we estimated the number of messages needed in each root-migration. If the short-message method is used in the minimum outgoing edge identification, the total number of messages needed in each root-migration will be at most $5|M|$, as proved in Gallager's.[7] If the long-message method is employed, the number of messages needed will be reduced.

The dependency between the upper bound of $|M|$ and the number of multi-hop root migrations is shown in Lemma 2.3.

Lemma 2.3: *The number of nodes in any fragment M at ith multi-hop root-migration in an execution of Algorithm DMST is less than or equal to $(n-1)-2(R_m-i)$, where R_m is the total number of multi-hop root-migrations in the execution.*

Proof: As shown in Lemma 2.2, the last node to be included into M must not be a blue node. Following the argument in Lemma 2.2, there must have $2(R_m-i)$ more nodes not in M for the other (R_m-i) multi-hop root-migrations.

□

Following the same argument, the upper bound of the number of yellow edges in each root-migration is obtained in Corollary 2.3. This is useful to estimate the message length.

Corollary 2.3: *The maximum number of yellow nodes in any 'Root-migration' message is at most $(n-1)-(R-i)$ or $(n-1)-2(R_m-i)$, where R is the total number of root-migrations and R_m is the total number of multi-hop root-migrations in the execution.*

The message complexity when the long-message method is used is now shown in Theorem 2.

Theorem 2: *Algorithm DMST-NJT needs at most $(2e + n(n-1)/4)$ messages if the long-message method is used in the minimum weight outgoing edge identification.*

Proof:

(a). It needs $2e$ messages for 'Follow-me' and replying messages as described before.

(b). In the worst case, that there are at most $(\lceil n/2\rceil-1)$ multi-hop root-migrations and no back-to-back root-migrations. Each multi-hop 'Root-migration' message travels a distance at most $|M|-1$. Furthermore, each multi-hop migration will include a new green node and a blue node into M. The number of hops the 'Root-migration' message travels in the ith iteration will be at most $|M|-1 \le ((n-1)-2(\lceil n/2\rceil-1-i))$. Let m_i to be $(n-2)-2(\lceil n/2\rceil-1-i)$, the total number of messages in root-migrations is at most

$$\sum_{i=1}^{\lceil n/2\rceil-1} m_i \le \sum_{i=1}^{\lceil n/2\rceil-1} (2i-1) \le (\lceil n/2\rceil-1)(n-1)/2 \le n(n-1)/4.$$

□

From the proof of Theorem 2, it can be easily seen that the length of a 'Root-migration' message is not a problem if the number of multi-hop root-migration is high. In the worst case, in which the number of root-migration is $(\lceil n/2\rceil-1)$, at most one yellow edge is added to M, and one yellow edge is changed to blue in each root-migration. Thus, the number of yellow nodes needed to be carried in each 'Root-migration' message is at most two. Although the message length may be longer in other cases, the number of 'Root-migration' messages and the distance to travel will be much smaller.

In the average case, the number of remote root-migrations will be less than that in the worst case $(\lceil n/2\rceil-1)$, since the average size of a fragment is usually greater than one and the number of hops for a 'Root-migration' message to travel is far less than $|M|$. Therefore, the 'Follow-me' and 'reply' messages will dominate the overall communication overhead in the average case.

4.3. Time Complexity

To discuss the time complexity, we differentiate between the time for node drafting and the time for root-migrations. The root drafting is a concurrent process, each node receives at most one 'Follow-me' message and replies one message at most. Thus, it needs $2n$ steps at most. On the other hand, root-migration is a sequential process. So the time complexity is the same as the message complexity in the worst case, which is $O(n^2)$.

In the best case when there is no root-migrations and the graph is not skewed, (that is, the diameter of the graph is not close to n), it will need only $2\log n$ steps. The time complexity in the best case will be in the order of $O(\log n)$.

4.4 Average Case Performance

Because it is extremely difficult to estimate the performance of the algorithm in the majority of cases, average performance is analyzed using simulation study. For each given number of nodes, we simulate the algorithm on 20 random regular graphs of various degrees and randomly select 10 different initial root nodes. (Whenever nd is odd, one node may not have the same degree.) Simulation results show that the number of messages is less than the Tree-join-tree algorithms when either long-message or short-message minimum outgoing edge identification methods are used. Table 1 shows the summary of simulation results. RM is the average number root-migrations, and U is

$$\sum_{\text{all iterations}} \text{number of green nodes},$$

which can be used to estimate the upper bound of the messages needed in both short-message method and long-message method. When the short-message method is used, the number of messages required is at most $2e + 5U$; while in the long-message method is at most $2e + U$. The simulation results are detailed in.[9]