

IMPROVING THE TIME COMPLEXITY OF MESSAGE-OPTIMAL DISTRIBUTED ALGORITHMS FOR MINIMUM-WEIGHT SPANNING TREES*

F. CHIN† AND H. F. TING‡

Abstract. A distributed algorithm is presented that constructs the minimum-weight spanning tree of an undirected connected graph with distinct node identities. Initially, each node knows only the weight of each of its adjacent edges. When the algorithm terminates, each node knows which of its adjacent edges are edges of the tree. For a graph with n nodes and e edges, the total number of messages required by this algorithm is at most $5n \log n + 2e$, where each message contains at most one edge weight plus $3 + \log n$ bits. Although the algorithm presented here has the same message complexity as the previously known algorithm due to Gallager, Humblet, and Spira [*ACM Trans. Programming Language and Systems*, 5 (1983), pp. 66–77], the time complexity of the algorithm presented improves from Gallager's $O(n \log n)$ to $O(n \log^* n)$ time units, where $\log^* k$ is the number of times the log function must be applied to k to obtain a result less than or equal to one. A worst case of $\Omega(n \log^* n)$ is also possible. In addition, when the network is synchronous, the algorithm presented is modified further to solve the same problem with the same message complexity but in $O(n)$ time.

Key words. distributed algorithms, synchronous and asynchronous networks, minimum spanning trees, communication complexity

AMS(MOS) subject classifications. 68M10, 68Q25

1. Introduction. Given an undirected connected graph G with n nodes and e edges, where each node has a unique identity, a **spanning tree** of G is a connected subgraph of G with exactly n nodes and $n - 1$ edges. The **weight of a spanning tree** is the sum of weights of all edges in the spanning tree. Our problem is to design a distributed algorithm that finds a spanning tree of G whose weight is minimum, i.e., the **minimum-weight spanning tree** (MST) of G .

We assume that a processor exists at each node of the graph, and the processor initially knows the weights of the edges adjacent to the node. Each node performs the same local algorithm and two adjacent nodes can communicate with each other by exchanging messages on the edge between them. A node can send (broadcast) or receive messages on several adjacent edges simultaneously. Messages can be transmitted independently in both directions on an edge, without error and in sequence. In an asynchronous network, each message sent by a node to any of its neighbors arrives within some finite but unpredictable time. However, in the synchronous network, there is a global clock accessible by all nodes, and messages are allowed to be sent only at integer pulses of the clock. During each clock pulse at most one message can be sent over a given edge and the delay of each message is at most one time unit (i.e., one pulse duration) of the global clock.

The **time complexity of a synchronous algorithm** is defined as the maximum number of clock pulses passed between the sending of the first and the receiving of the last message of the algorithm. In an asynchronous network, we assume the existence of a hypothetical global clock. The processing and queueing time of each message is negligible, and the transmission of each message takes at most one time unit. The **time**

* Received by the editors January 20, 1987; accepted for publication (in revised form) September 15, 1989.

† Department of Computer Science, University of Hong Kong, Pokfulam, Hong Kong.

‡ Department of Computer Science, Princeton University, Princeton, New Jersey 08544.

complexity of an asynchronous algorithm is the maximum number of time units from the start to the completion of the algorithm. This assumption is introduced only for the purpose of performance evaluation and the algorithm can operate correctly with arbitrary delays.

Since Minimum-Weight Spanning Tree (MST) is one of the most fundamental structures of a graph, it is not surprising that the MST construction algorithm can serve as building block for many other distributed algorithms, such as network synchronization [2], breadth-first-search [3] and deadlock resolution [4]. Awerbuch has proved in [5] that this problem is equivalent to a large class of problems (e.g., leader selection, spanning tree construction, counting the number of nodes in a network and computing a sensitive decomposable function).

To the best of our knowledge, all distributed algorithms that solve the MST problem [5], [11], [12], [15], [18] employ the idea of Borůvka [7], [8], [14], the so-called Sollin algorithm. A **fragment** is defined to be a subtree of a MST. Initially each node is treated as a fragment, and fragments are merged together iteratively over their minimum-weight outgoing edges. An edge e is an **outgoing edge** of a fragment if one of its endpoints is in the fragment and the other is not. The algorithm terminates when one fragment remains. Each fragment finds its minimum-weight outgoing edge independently and has a designated edge called the **core** to coordinate action. The algorithm proceeds in phases in which fragments are merged into larger ones. During each phase, information must be broadcast from the core to every node in the fragment and vice versa. Unfortunately, the message complexity of the obvious implementation of this algorithm is $\theta(n^2)$. Message complexity is worst when there is a large fragment (say a fragment with $n/2$ nodes) that goes through $n/2$ phases by enlarging its size one node at a time. Since during each phase at least $n/2$ messages are needed for the communication between the core and all other nodes, $\theta(n^2)$ messages are needed for the algorithm.

Gallager, Humblet, and Spira [13] later proposed an improved algorithm that solves this problem in $\theta(e + n \log n)$ messages. When two fragments are merged, Gallager's algorithm ensures that the small fragment is merged into the large one and work is only done by the small fragment. Since each fragment is always merged into a fragment of at least its original size, each node can go through at most $\log n$ merges. To implement this mechanism, a **level** field is associated with each fragment, with the property that a fragment at level l has at least 2^l nodes; in particular, fragments with a single node have $l=0$. The level of a node is defined as the level of its containing fragment. Thus, whenever two fragments at different levels join, the fragment at the smaller level works harder, never waits, and assumes the name and level of the larger. On the other hand, if a higher level fragment tries to merge with a lower level one, the former fragment waits until the latter fragment reaches a level high enough for combination. When two fragments at the same level join, both fragments go through the name change and a new fragment with its level increased by one is formed.

As given in [13], $\Omega(e)$ messages are necessary in constructing a spanning tree, and as proved in [16], $\theta(n \log n)$ messages are needed to find a leader on a ring. It follows that $\Omega(e + n \log n)$ messages are necessary to construct a spanning tree in a general network. Thus, Gallager's algorithm is message-optimal. However its worst-case time complexity is $\theta(n \log n)$. Let us consider a fragment of size $n/2$ but at a low level, say $l=1$. Because of its initially low level, this fragment can go through $\log n$ merges before forming the final MST. Since each merge may take $\theta(n)$ time to update the information of the fragment, Gallager's algorithm might require $\theta(n \log n)$ time. This happens because the size of a fragment is not reflected by its level. To be more

specific, when one fragment joins another, high-level fragments must wait for low-level ones to respond. But the size of those low-level fragments may be large and messages may have to travel from one end of the fragment to the other end, in order to update the level and identity of the fragment and to find its minimum-weight outgoing edge. Thus, the waiting time can be unduly long.

In this paper, we modify Gallager's algorithm by introducing the property that a fragment at level l has size bounded within 2^l and 2^{l+1} , otherwise its level will be updated. Thus, if there exists a large fragment, its level will be increased accordingly and its workload can therefore be reduced. The message complexity of this algorithm remains $5n \log n + 2e$ but the time complexity can be reduced to $15n \log^* n + 3n$. The detailed description of our algorithm is given in § 2. In fact, similar observations and results have also been obtained independently by Gafni [11]. In § 2.4, an example that uses $\Omega(n \log^* n)$ time is presented.

In § 3, we further modify the algorithm for a synchronous network so that a high-level fragment can immediately terminate its waiting for another low-level fragment's response if the waiting time of the high-level fragment is unduly long. Should this happen, the size of these low-level fragments must be large enough for merging and thus the waiting of the high-level fragment can be terminated and be merged with the low-level fragment correctly. Since the durations of all waitings that depend on the levels of the corresponding fragments are bounded, the algorithm can be executed in synchronized phases and terminates in $\theta(n)$ time. Recently, Awerbuch [5] has shown that the asynchronous MST problem can also be solved in $\theta(n)$ time with $O(e + n \log n)$ messages.

2. The asynchronous algorithm. Since our algorithm depends very much on the one given by Gallager, Humblet, and Spira [13], we adopt similar notation in our algorithm description. In particular, uppercase words stand for labels and states, whereas italic words are for messages. Our algorithm starts with all n nodes awake. This assumption can be relaxed by propagating an *AWAKE* message to all nodes initially. This can be done with at most $n - 1$ extra time units and $2e$ extra messages. Initially, each node is a fragment at level zero. Fragments are merged together iteratively as described in Gallager's algorithm. However, the method for determining a fragment level and the mechanism for merging fragments are somewhat different.

As given in [13], we have the following definitions and properties:

(1) A node has two possible states: **FIND** and **FOUND**. Initially, all nodes are in the **FOUND** state.

(2) An edge has three possible states. Initially, all edges are in the **BASIC** state. An edge is in the **SELECTED** state if it is found to be a MST edge. It is in the **REJECTED** state if it is known not to be a MST edge.

(3) Every fragment usually has a **SELECTED** edge as its core. The adjacent nodes of the core act as the coordinators. For those fragments that do not have a core, a node is designated as the coordinator.

(4) Every fragment has a fragment level and a **fragment identity**, which equals to the weight of either its core or an edge incident to its coordinator (if there is no core in the fragment).

2.1. How a fragment finds its minimum-weight outgoing edge. Whenever a new fragment is formed, the coordinator(s) of the fragment broadcasts the message *INITIATE*(F, l, FIND) along the **SELECTED** edges of the fragment. As this set of **SELECTED** edges forms a spanning tree of the fragment, all the nodes in the fragment are

informed about their new fragment identity F and their new fragment level l . At the same time, all the nodes change their states to **FIND** in order to participate in finding the fragment's minimum-weight outgoing edge.

When a node u enters the **FIND** state, it finds the minimum-weight outgoing edge by sending a $TEST(F, l)$, message over its minimum-weight **BASIC** edge to, say, node u' in fragment F' at level l' , and waits for a response. If the response message is **REJECT**, i.e., F and F' have the same identity, then u marks the edge **REJECTED** and sends a $TEST$ message over its next minimum-weight **BASIC** edge. The $TEST$ message will be sent until either an **ACCEPT** message is received or there are no more **BASIC** edges adjacent to u . If u receives an **ACCEPT** message on a **BASIC** edge, there it will remember the edge as **min_edge** and its weight as **min_weight**. On the other hand, if there is no outgoing edge from u , **min_weight** is set to infinity.

When a node u' in fragment F' at level l' receives a $TEST(F, l)$ message on a **BASIC** edge, node u' responds with a **REJECT** message and marks the edge as **REJECTED** if F and F' are equal. If F and F' are different and $l \leq l'$, then u' responds to u immediately with an **ACCEPT** message. On the other hand, if $l > l'$, u' will wait until $l' \geq l$.

A node u in state **FIND** will eventually send a $REPORT(W, SZ)$ message along the **SELECTED** edges to its coordinator, where W stands for the weight of the minimum-weight outgoing edge and SZ for the size of the subfragment root at u . Assume W_i and SZ_i are the minimum-weight outgoing edge and the size of the subfragment rooted at u 's i th son. If u is a leaf node, $W = u$'s **min_weight** and $SZ = 1$, otherwise, $W = \text{the minimum of } \min(W_i) \text{ and } u\text{'s min_weight}$ and $SZ = 1 + \sum SZ_i$. At the same time, node u marks the minimum-weight outgoing edge of the subfragment rooted at itself, i.e., its **min_edge** or the edge leading to its son that has the minimum W_i . Node u sends a $REPORT(W, SZ)$ message to its father and changes its state to **FOUND** only after it has found its **min_weight** and has received all $REPORT(W_i, SZ_i)$ messages from its sons (if any). When the coordinator(s) of a fragment receives all **REPORT** messages from its sons, the fragment size and the weight of the minimum-weight outgoing edge can be determined.

The algorithm terminates when the returned value W in the **REPORT** message at the coordinator is infinity. This implies there are no outgoing edges and there is only one fragment in the graph.

As shown in [13], when the coordinators receive all the **REPORT** messages, it must be the case that $SZ \geq 2^l$ where l is the fragment level. But since the time the fragment initiated the process of finding its minimum-weight outgoing edge, other fragments may have merged into it and its fragment size and level may not be commensurate, i.e., $SZ \geq 2^{l+1}$. As we must make sure that the fragment level always reflects its size, the coordinator(s) compares SZ with 2^{l+1} . If $SZ \geq 2^{l+1}$, the coordinator(s) broadcasts another **INITIATE**(F, l', FIND) message to all nodes in the fragment to update their level as if a new fragment at level l' has just formed, where l' is the largest integer such that $SZ \geq 2^{l'}$. Note that this level-updating process may be repeated many times until the fragment size and level are commensurate (i.e., $2^l \leq SZ < 2^{l+1}$). On the other hand, if $SZ < 2^{l+1}$, then the fragment is ready to combine with another fragment to form a new fragment with a new core. The coordinator(s) sends a message **CHANGECORE** following the path of the *marked* edges to the node adjacent to the minimum-weight outgoing edge.

2.2. How fragments are merged together. When the **CHANGECORE** message in a fragment at level l reaches node u , to which the minimum-weight outgoing edge

(u, u') is incident, u attempts to merge its fragment with the fragment F' at level l' that contains u' by sending a *CONNECT*(l) message over (u, u') . After receiving this *CONNECT*(l) message, u' compares l with its fragment level l' . There are two possible outcomes, $l = l'$ or $l < l'$. The outcome $l > l'$ is not possible because levels are nondecreasing and a *TEST* message must have been sent and responded to before this *CONNECT*(l) message is sent.

Case 1. $l = l'$. If u' has previously sent a *CONNECT*(l') message over edge (u, u') to u , then the two fragments will have the same minimum-weight outgoing edge, and will be merged immediately together to form a new fragment F'' at level $l+1$ with edge (u, u') as its new core. Edge (u, u') is marked *SELECTED*. *INITIATE*(F'' , $l+1$, *FIND*) messages are then broadcast to all nodes in F'' . In all other cases, u' will wait until it sends a *CONNECT*(l') message to u and proceeds as previously described, or until it increases its level l' , in which case it does the following.

Case 2. $l < l'$. Normally, fragment F is merged with fragment F' . Because of our strategy of never making a low-level fragment wait, node u' immediately marks edge (u, u') as *SELECTED* and sends an *INITIATE*(F' , l' , S') message to u , where F' , l' , and S' stand for fragment identity, level, and state, respectively, of u' .

If $S' = \text{FIND}$, then u' has not sent its *REPORT* message. Fragment F simply joins fragment F' and participates in finding the minimum-weight outgoing edge of the enlarged fragment. Node u marks edge (u, u') as *SELECTED*, changes its state to *FIND*, fragment identity to F' , fragment level to l' , and also relays the *INITIATE*(F' , l' , S') message to all other nodes in F . Meanwhile, u' waits for the *REPORT* message from u before sending its *REPORT* message.

If $S' = \text{FOUND}$, then u' has already sent its *REPORT* message. Thus the size of fragment F cannot be reported to the coordinator of F' . However, we want to make sure that its size is reflected at the next level update. Under such circumstances, fragment F will not be merged with F' immediately but instead will be treated as a new fragment with a new identity and a new level. Basically, node u will be the new coordinator of the fragment. It changes its fragment identity to ω , the weight of a *SELECTED* edge incident to u , its state to *FOUND*, its level to l' , broadcasts an *INITIATE*(F , l' , *FOUND*) message with $F = \omega$ to all the nodes in the fragment, and waits for their *REPORT* messages. As edge (u, u') remains the minimum-weight outgoing edge for this new fragment, state *FOUND* is assigned to all nodes in the fragment. Note that u will not mark edge (u, u') as *SELECTED*, whereas on the other hand, u' has already sent its *REPORT* message and has marked edge (u, u') as *SELECTED*. Thus, messages can still be transmitted from u' to u as if they are in the same fragment, so it is possible for u to receive another *INITIATE* message from u' before u receives all its sons' *REPORT* messages. In order to ensure that every *INITIATE* message has been reported before another *INITIATE* message is issued, node u may have to delay its action on the second *INITIATE* message until it has received all the *REPORT* messages from its first *INITIATE* message. Hence, there is at most one such pending *INITIATE* message from u' .

Having received all its sons' *REPORT* messages, u compares its fragment size SZ with $2^{l'+1}$. F and F' are combined together only when $SZ < 2^{l'+1}$. In other words, when the size of F is small enough and is reflected by its new level, the size of F does not have to be reported to the coordinator of F' and F can be merged into F' without problem. On the other hand, if the size of F is sufficiently large, F will not be combined with F' . In order to prevent a large fragment from merging with a small one, we delay the process in F and make F wait by increasing its level sufficiently. Thus there are two cases:

(a) $SZ < l'^{+1}$. Fragment F can be absorbed into fragment F' . Node u will mark edge (u, u') as **SELECTED**. If there is a waiting **INITIATE** message, u will process the second **INITIATE** message as if it had just been received. Even though F and F' have different identities in this merged fragment for an uncertain period before this second **INITIATE** message is processed in F , this does not create any problems in checking whether an edge is an outgoing edge from F' to F . This is because if a **TEST** message is sent over an edge, say (ν', ν) from F' to F , then the fragment level of ν will be less than that in the **TEST** message (i.e., the fragment level of ν'). Node ν would delay making any response until it receives the second **INITIATE** message and obtains the same fragment identity and level as node ν' . Thus, that **TEST** message will be rejected eventually.

(b) $SZ \geq 2^{l'+1}$. Let l'' be the largest integer such that $SZ \geq 2^{l''}$. Node u will change its level to l'' , its state to **FIND**, and will send a **REPORT**($w, 0$) to u' , where w is the weight of the edge (u, u') . Furthermore, node u will broadcast the **INITIATE**(F, l'', FIND) to all its sons as if a new fragment has just been formed.¹ On receiving a **REPORT**(w, x) message with $x=0$ over a **SELECTED** edge, node u' remarks that edge as **BASIC** and handles the **REPORT** message as usual. From then on, fragment F and F' are treated as separate fragments.

2.3. Analysis of the algorithm. The correctness proof and message complexity analysis are same as given in Gallager, Humblet, and Spira [13]. For the correctness proof, we only need to prove that in our modified algorithm, deadlock will not be created and an edge is a branch of the MST if and only if it is **SELECTED**. From the description of our algorithm in the previous section, our only modifications are to raise the level of a fragment when its size is too large and to delay the merging of the fragments when its resultant size cannot be reflected by its level. However, these modifications would not change the fact that only the minimum-weight outgoing edge of a fragment will be marked **SELECTED** and that the smallest-level fragments never wait. With the same argument as in [13], fragments remain to be subtrees of the MST and there is no deadlock in the algorithm.

As far as the message complexity is concerned, it seems that more messages are required to ensure the size of each fragment is reflected by its level. However, this increase of messages is always associated with a level change of a fragment and each node at any given level still transmits/receives at most five messages [13]. Since fragments are always merged into larger fragments and their levels only increase, a node can go through at most $\log n$ levels and this accounts for the $O(n \log n)$ messages. With the fact that an edge can be rejected only once, and each rejection requires two messages, there are at most $2e$ messages leading to rejections. Thus, the total message complexity remains $O(e + n \log n)$.

In order to prove the upper bound on time complexity, we have the following definitions. A **fragment** F is the largest minimum-spanning subtree whose vertices have the fragment identity F . A **subfragment** of F is a fragment whose next fragment identity is F . If F_1, F_2, \dots, F_m are all the subfragments of F , it can be shown easily that all F_i 's are disjoint, $F = \bigcup_{i=1}^m F_i$, and $SZ = \sum_{i=1}^m SZ_i$, where SZ and SZ_i stand for the sizes of the fragment F and F_i , respectively.

For $0 \leq i \leq \log n$, let τ_i be the minimum time of the hypothetical global clock at which all nodes in the graph have level at least i and $a_i = \tau_i - \tau_{i-1}$. A **critical node** at

¹ We now briefly describe a modification of the algorithm that is used to improve its complexity. If the waiting **INITIATE** message is of level at least l'' , then it can be processed immediately without broadcasting the **INITIATE**(F, l'', FIND) message to all the nodes in F .

τ_i is the node that changes from a level less than i to a level at least i at time τ_i . A **critical fragment** at τ_i is the fragment containing a critical node at τ_i at time τ_i . Note that there may be several critical nodes at τ_i , and likewise, several critical fragments. Since all the critical fragments are disjoint and the same argument can be applied to them, without loss of generality, let us consider a particular critical node and the corresponding critical fragment. An **active node set**, AN_i , at τ_i is the set of nodes in the critical fragment at τ_i with level less than i at time τ_{i-1} . From the definition of τ_{i-1} , these nodes are at level $i-1$. If the active node set at τ_i is nonempty, this set of nodes may belong to several subfragments at level $i-1$ of the critical fragment at τ_i . As all nodes are initially AWAKE, $\tau_0 = 0$. If l is the fragment level when the algorithm terminates, then $\tau_i = \tau_l$ for $l \leq i \leq \log n$. Define,

$$\log^{(k)} n = \begin{cases} \log n & \text{if } k = 1, \\ \log(\log^{(k-1)} n) & \text{if } k > 1. \end{cases}$$

It is obvious that $\tau_{\log n} = \sum_{i=1}^{\log n} a_i$. We now partition the indices i into sets. Let $S(k)$ be the index set $\{i: (\log n - \log^{(k)} n) < i \leq (\log n - \log^{(k+1)} n)\}$. For example, $S(1) = \{1, 2, \dots, \log n - \log^{(2)} n\}$, $S(\log^* n - 2) = \{\log n - 3, \log n - 2\}$, $S(\log^* n - 1) = \{\log n - 1\}$ and $S(\log^* n) = \{\log n\}$. The a_i 's are partitioned into classes according to these index sets, such that all the a_i 's whose indices belong to the same index set are in the same class. We want to show that the sum of all a_i 's in any class is $O(n)$. Since there are $\log^* n$ classes in all, we have $\tau_{\log n} = O(n \log^* n)$.

Let us consider the k th class. The elements i in $S(k)$ are partitioned into two groups, $S'(k)$ and $S''(k)$, according to the value of the corresponding a_i 's. $S'(k)$ contains the indices of all the small a_i 's and $S''(k)$ contains the indices of the large ones; more precisely, $S'(k) = \{i \in S(k): a_i \leq 10n/\log^{(k)} n\}$ and $S''(k) = S(k) - S'(k)$. Since $S(k)$ contains $(\log^{(k)} n - \log^{(k+1)} n)$ elements and $S'(k) \subseteq S(k)$, the sum of all the small a_i 's whose indices are in $S'(k)$ is no more than $O(n)$. As for the large a_i 's corresponding to $S''(k)$, we will prove that the sizes of their corresponding AN_i 's, and their corresponding critical fragments, cannot be small. Intuitively, this can be seen as follows. If the AN_i corresponding to a large a_i is small, then the time required to change the level of these nodes in AN_i 's is small, contradicting the fact that a_i is large. Since our algorithm makes sure that sizes of fragments are reflected by their levels, the level of each fragment containing a large AN_i will be increased accordingly. As AN_i is large, so is its level increase. Thus, the level of the nodes in these large AN_i 's becomes so high that these nodes cannot be in another AN_j in the same class as a_i . Thus, there cannot be too many large a_i 's in the k th class corresponding to $S''(k)$, and consequently the sum of the large a_i 's whose indices are in $S''(k)$ is also $O(n)$. We now make these observations precise.

LEMMA 1. $a_i \leq 5|AN_i|$, where a_i is the time required to increase the level of all nodes with level less than i at τ_{i-1} , which includes the active node set AN_i , to a level at least i .

Proof. The lemma holds true for $a_i = 0$. Assuming $a_i > 0$, we have $\tau_i > \tau_{i-1}$ and $AN_i \neq \emptyset$. At time τ_{i-1} , every node v in AN_i has received the *INITIATE* message at level $i-1$ and is ready to send or has sent some *TEST* messages. Let us consider the situation when v must send some *TEST* messages to find its *min_weight*. All nodes in the fragment to which v belongs at time τ_{i-1} must be at level $i-1$, and the size of this fragment is at most $|AN_i|$. Hence, v will send at most $|AN_i|$ *TEST* messages and will receive at most $|AN_i|-1$ *REJECT* messages before an *ACCEPT* message is received. As v 's level is the lowest in the graph, the *TEST* messages will be responded to without any delay, and thus after at most $2|AN_i|$ time units, all the leaf nodes in AN_i relative to the tree corresponding to the critical fragment can report. As there are

$|AN_i|$ active nodes, there will be at most $|AN_i|$ *REPORT* messages, $|AN_i|$ *CHANGECORE/CONNECT* messages, and $|AN_i|$ *INITIATE* messages from within the critical fragment. Thus, it takes at most $3|AN_i|$ extra time units before all nodes in AN_i rise to a level at least i . Thus, we have $a_i \leq 5|AN_i|$. \square

COROLLARY 1. *If $a_i > 5m$, then $|AN_i| > m$, where m is any positive integer.*

LEMMA 2. *Let v be a node in fragment F at level l at time t . On its next level update, v will change its level from l to l' such that $2^{l'+1} > |F|$.*

Proof. The proof follows directly from the description of the algorithm. The size of the fragment must be reported after every *INITIATE* message, which is the only way to change the level of a fragment. The new level is assigned according to the fragment size. \square

Let $R(k) = \sum_{i \in S(k)} a_i$ for $1 \leq k \leq \log^* n$. Then $\tau_{\log n} = \sum_{k=1}^{\log^* n} R(k)$.

LEMMA 3. $R(k) \geq 15n$ for $1 \leq k \leq \log^* n$.

Proof. $S(k)$ is partitioned into $S'(k)$ and $S''(k)$. $R(k) = R'(k) + R''(k)$ where

$$R'(k) = \sum_{i \in S'(k)} a_i \quad \text{and} \quad R''(k) = \sum_{i \in S''(k)} a_i.$$

Since $a_i \leq 10n/\log^{(k)} n$ for all $i \in S'(k)$, we have

$$R'(k) = \sum_{i \in S'(k)} a_i \leq \left(\frac{10n}{\log^{(k)} n} \right) (\log^{(k)} n - \log^{(k+1)} n) \leq 10n.$$

As $a_i > 10n/\log^{(k)} n$ for all $i \in S''(k)$, from Corollary 1 we have $|AN_i| > 2n/\log^{(k)} n$. The initial level of a new fragment may not reflect its size, but by Lemma 2, all nodes in AN_i will raise their level to greater than $\log n - \log^{(k+1)} n$ at their next level update. From then on, by the definition of $S(k)$, these nodes will never belong to any other active node set corresponding to $S(k)$, i.e., they can only belong to some active node set AN_j in some $S(k')$, with $k' > k$. Thus, before this level update, all these nodes in the graph may belong to at most one active node set corresponding to $S''(k)$. Using this fact and Lemma 1, we have $R''(k) = \sum_{i \in S''(k)} a_i \leq \sum_{i \in S''(k)} 5|AN_i| \leq 5n$. Hence $R(k) = R'(k) + R''(k) \leq 15n$. \square

Using Lemma 3 and the equality $\tau_{\log n} = \sum_{k=1}^{\log^* n} R(k)$, we have the following theorem.

THEOREM 1. $\tau_{\log n} \leq 15n \log^* n$. \square

After $\tau_{\log n}$ and before the algorithm terminates, at most $3n$ time units are required for the *TEST*, *REJECT*, and *REPORT* messages as described in the proof of Lemma 1. Thus, our algorithm takes no more than $15n \log^* n + 3n$ time units.

2.4. Example with time complexity $\Omega(n \log^* n)$. Let us consider two extreme situations. If there are many small fragments merging in pairs, each node may participate in $\log n$ level changes before the algorithm terminates. However, under this situation, our algorithm allows a large amount of parallelism in message exchanges and each fragment doubles its size at each merging or level change. Since the time delay for each level change is proportional to the fragment size, the total time delay of the algorithm would still be $O(n)$. If, on the other hand, there is a large fragment, our algorithm guarantees that the fragment's level will be raised appropriately. Even though there might be a long delay for this level update, this time loss can be compensated by the large level increase and the algorithm can still be finished in $O(n)$ time. We can show that the worst time complexity of our algorithm occurs when there is a sequential waiting of the fragments with size $n/\log n$, and after $O(n)$ time, the number of fragments can at most be reduced from n to $\log n$. This process can be repeated for $\log^* n$ times until a single fragment remains. Here is a sketch of our example whose

execution requires at least $\log^* n$ stages with each stage takes $\Omega(n)$ time. Thus, the whole execution of our example requires $\Omega(n \log^* n)$ time.

Let us first consider $\log k$ rows of nodes, each row has k nodes, connected in a line as shown in Fig. 1, which is an example with $k = 8$. The nodes in each row are further subdivided into groups; the i th row is divided into $k/2^{i-1}$ groups, each with 2^{i-1} nodes (the nodes in each oval form a group). The nodes in each group are linked together with edges of lowest weight, and thus they will first merge together to form supernodes of level $i-1$ (for nodes in the i th row). Furthermore, these groups are connected in a line with edge weights strictly ascending from left to right. When the minimum-weight spanning tree algorithm is executed, the nodes in the ovals form supernodes in each row. Let us consider the supernodes of the i th row where $i > 1$. They are at level $i-1$ and their leftmost supernode sends a *TEST* message over an edge of the highest weight (the 1,000 link in Fig. 1) to a supernode of the $(i-1)$ st row at level $i-2$. This *TEST* message will be responded to when all the nodes in the $(i-1)$ st row have merged into a single fragment with its level increased to $\log k$. Then, the leftmost two supernodes in the i th row will merge together to form the core of a new fragment. The other supernodes in the same row will then merge one after another without concurrency in a strictly left to right fashion. Finally, all the nodes in each row will merge into a fragment of level $\log k$, with the leftmost intersupernode edge as the core.

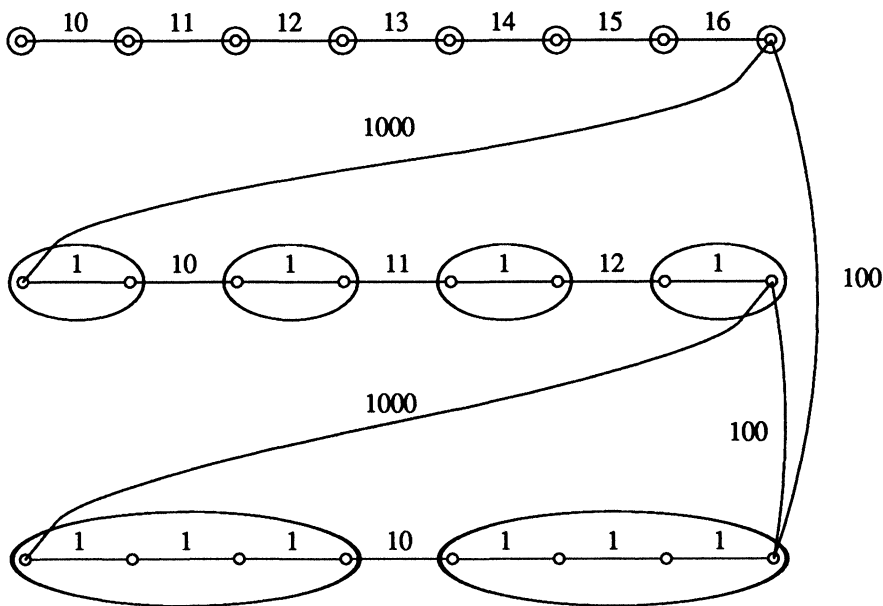


FIG. 1

Although each row will eventually form a single fragment, this cannot happen until all the nodes in the preceding row have merged together and have had their levels updated to $\log k$. Since there are $\log k$ rows and the level update of all the nodes in each row takes $\theta(k)$ time units, it takes $\theta(k \log k)$ time for all the rows to become fragments. If we let k be about $n/\log n$, then it takes $\theta(n)$ time for all the rows to form single fragments.

Let F_i be the fragment formed by the i th row. F_i is at level $\log k$ and, except for the last row, sends a *TEST* message over the edge of the second highest weight (the 100 links in Fig. 1) to a node at the last row which is at level $(\log k - 1)$. Since that *TEST* message goes to a fragment at a lower level than F_i , F_i must wait until the last row has formed a single fragment before connecting to another fragment. Basically, these second highest weight edges (the 100 links) are for the purpose of synchronizing each stage.

The above construction is repeated for several stages by treating each F_i as a supernode. At each stage, the number of supernodes is the logarithm of the number of nodes (supernodes) in the preceding stage. So, this whole process will be done in $\Omega(\log^* n)$ stages and it will take $\Omega(n \log^* n)$ time to complete the entire algorithm. In order that the execution of each stage is not affected by what is done in the preceding stages, without loss of generality, let us consider the second stage. Two new nodes a , b are introduced for each fragment F_i , and they are connected with edges of weight x , which is larger than those on any of the edges of but less than those edges between the F_i 's, as in the Fig. 2.

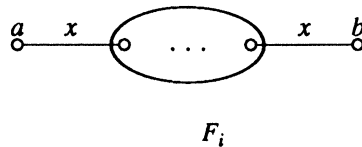


FIG. 2

Let the two new nodes a , b be the **left** and **right handles** of F_i , respectively. By making the edges connected to the handles are of weight larger than x , the handle will join the subfragments of F_i at an early stage, and will have no effect on the preceding stage. In addition, any edges attached to the handles will not affect the execution of the preceding stage either. In other words, F_i is formed in the first stage even with the handles. Fragments F_i 's are then connected into the same structure as in Fig. 1 by treating each F_i , together with its handles, as a supernode.

With k , the number of nodes (fragments) in each row, equals to $\log n / \log \log n$, there will be enough fragments F_i . The supernodes are then connected in such a way that edges coming into the left of the supernode are attached to its left handle, and those connected to the right are connected to the right handle to make sure that the second stage could not start until the first stage is complete. The weights of the new edges will all be larger than x and in the mid-range (say 50, if Fig. 1 is used as an indicator of weights in the preceding stage). That is, they will be larger than the weights of the branches within each F_i , and they will be less than the previous stage's edges between the F_i .

We start in the first stage the construction with k nodes in each row and $\log k$ rows. With $k = n / \log n$, there are about $(n / \log n)(\log n - \log \log n)$ nodes. Since each fragment F_i requires a pair of handles, it requires at most $2 \log n$ handles at the first stage. Similarly, $2 \log \log n$ for the second stage. Thus, there are a total of no more than $2(\log n + \log \log n + \dots)$ nodes added. So for large n , the number of nodes in the final graph is less than n . Note that the graph can be padded out to just n nodes in any reasonable fashion.

3. The synchronous algorithm. Let us consider the above algorithm in more detail and understand why the nonlinear time bound still exists. The worst-case time complexity occurs when there is sequential waiting of medium-size fragments. The nonlinear characteristic of the time complexity is due to the fact that the level of a fragment

cannot be increased gradually step by step. If many mergings to fragment F occur almost at the same time, there may be a large level increase of fragment F , say from l to l' , after 2^l time units. In the meantime before F changes its level to l' , another fragment F'' at a slightly higher level than F , say l'' , where $l < l'' \ll l'$, may have sent a *TEST* message to fragment F and is waiting for the level change of fragment F to l' in order to raise its own level from l'' to $l'' + 1$. Since the waiting can be unnecessarily long (up to 2^l time units) for one unit of level increase from l'' to $l'' + 1$, the time complexity of the algorithm is nonlinear.

In this section, we propose a synchronous version of our previous algorithm for solving this MST problem in $\theta(n)$ time with the same $\theta(e + n \log n)$ message complexity. This algorithm is a modification of the algorithm by Gallager, Humblet, and Spira [13] and tackles the above problem with an approach different from that of Awerbuch [5]. Awerbuch's approach is to reimburse fragment F'' for its time loss by hooking fragment F'' onto fragment F and subsequently inheriting level l' . In this case, fragment F'' has got its reward through its long waiting. Instead of hooking fragment F'' with fragment F through the minimum-weight outgoing edge as in the traditional methods, an arbitrary edge is chosen and a spanning tree is found instead of a minimum-weight one.

In order to see how our synchronous MST algorithm can be finished in $O(n)$ time, without loss of generality, let us assume that all the nodes awake simultaneously and execute the algorithm at the same time. Let T_l be the minimum time² when all the fragments in the graph have sizes at least 2^l (clearly, $T_0 = 0$). Thus, if a *TEST* message is sent after T_l from fragment F of level l through an interfragment edge to another fragment F' , the *TEST* message can be accepted immediately regardless of the current level of F' . On the other hand, if the *TEST* message is sent through an intrafragment edge and if fragment F is of size less than 2^{l+1} , all its node's identities and levels would have been updated by time $t_B + 2^{l+1}$, where t_B is the broadcast time of the *INITIATE* message by the coordinators and the *TEST* message would be rejected (assuming that the *TEST* message is sent after $t_B + 2^{l+1}$). If, however, the size of fragment $F \geq 2^{l+1}$, i.e., its size is not reflected by its level, the core will broadcast another *INITIATE* message and will command all the nodes to re-find the minimum-weight outgoing edge again. Based on this observation, if we modify our algorithm so that the *INITIATE* message is broadcast by the core only after T_l , i.e., $t_B > T_l$, and all the nodes participate in finding the minimum-weight outgoing edge after time $t_B + 2^{l+1}$, the *TEST* messages can be responded to immediately without considering the levels of the fragments, and the minimum-weight outgoing edge can be found correctly.

For every increase in fragment size, messages such as *INITIATE*, *REPORT*, and *CHANGE CORE* may have to traverse from one end of the fragment to another, but this takes at most $O(2^{l+1})$ time units. Since all messages are responded to without delay, the time T_{l+1} , which guarantees that all fragments are of size at least 2^{l+1} , can be bounded by the inequality $T_{l+1} \leq T_l + O(2^{l+1})$. Based on the recurrence formula for T_{l+1} , we can show that $T_{\log n} = O(\sum_{l=1}^{\log n} 2^l) = O(n)$, i.e., the time complexity of the algorithm is $O(n)$. We shall prove later that this recurrence inequality for T_{l+1} can also account for the fact that the nodes may not execute the algorithm simultaneously.

The algorithm can be described as follows. It starts by broadcasting *AWAKE* messages to all nodes across the whole network and any newly formed fragment starts

² Note that T_l is different from τ_l . In all cases $T_l \leq \tau_l$ because a fragment is formed before all the nodes in the fragment obtain their fragment level and identity.

to find its minimum-weight outgoing edge by broadcasting an *INITIATE*(F, l, S, t_B) message from its core, where the arguments stand for the fragment identity, fragment level, fragment state,³ and the broadcast time of the message. In order to guarantee the minimum size of fragments, we must have the broadcast time $t_B \geq t_F + T_l + 2^{l+1}$ where t_F is the wake-up time of the fragment F , i.e., the minimum wake-up time of its coordinator(s) and T_l will be defined later. Each node ν in the fragment then starts finding its own minimum-weight outgoing edge independently after time $t_B + 2^{l+1}$. This ensures that the process of finding the minimum-weight outgoing edge is synchronized among **all** nodes in the fragment.

Node ν sends a *TEST*(F) message through its minimum-weight adjacent BASIC edge and waits for a response in exactly two time units. Note that there is no level argument in the *TEST* message and the response is immediate without considering their levels. The responded message is *REJECT* when F and F' have the same identity, and *ACCEPT* otherwise. Immediately after the minimum-weight outgoing edge of a node is found, *REPORT*(W, SZ) messages are reported to the core. *CHANGECORE* and *CONNECT* messages are sent as in Gallager's algorithm.

Note that if the level of the fragment cannot reflect its reported size, another phase of *INITIATE* messages may be needed. However, when fragment F' at level l' receives a *CONNECT* message from fragment F at level l , this situation can be slightly different because the case $l > l'$ is possible. Should this happen, fragment F waits until fragment F' has reached a level high enough for combination in order to guarantee that the message complexity be bounded by $O(e + n \log n)$, and then F' sends an *INITIATE* message to F as described in the previous algorithms.

As a digression, Awerbuch [6] defined an **H-partition** problem which is to partition a given graph into disjoint fragments such that each of which has at least H nodes. The construction of an H -partition for a graph is useful in a number of applications [1], [17]. If our algorithm terminates at T_l for $l \leq \log n$, we can guarantee that all the fragments are of size at least 2^l . Thus, we can also solve the H -partition problem in $O(H)$ time and with $O(e + n \log H)$ messages as given in [6].

3.1. Correctness and complexity analysis. For proving the validity of our proposed synchronous algorithm, we must show in Lemma 4 that if T_l is defined as

$$T_{l+1} = \begin{cases} T_l + 28 \cdot 2^l, & \text{if } l \geq 1, \\ 0 & \text{otherwise,} \end{cases}$$

then any fragment F with size SZ satisfying $2^{l-1} \leq SZ < 2^l$ will correctly find its minimum-weight outgoing edge by time $t_F + T_l - 2^l$. Thus, we can guarantee that F would have merged into a larger fragment of size at least 2^l by time $t_F + T_l$.

LEMMA 4. *Any fragment F with size SZ satisfying $2^{l-1} \leq SZ < 2^l$ can correctly find its minimum-weight outgoing edge for merging before time $t_F + T_l - 2^l$, where t_F is the wake up time of fragment F .*

Proof. The proof is by induction on the level l . Initially, every node has size $SZ = 1$, thus the hypothesis is true for $l = 1$. For the induction step, assume that the hypothesis is true for $l \leq k$. Let us consider any fragment F , with size $2^k \leq SZ < 2^{k+1}$. We want to show that the sizes of all the subfragments of F are less than 2^k , and thus we can apply the induction hypothesis to show that the subfragments can find their minimum-weight outgoing edge and merge together to form F within a bounded period,

³ This state information FIND/FOUND can be omitted and this algorithm will also work with the assumption that each node is always in the FIND state.

i.e., $t_F + T_k + 2^{k+1}$. Since all the messages, such as *INITIATE*, *TEST*, *REJECT/ACCEPT*, *REPORT*, can be transmitted without delay and $SZ < 2^{k+1}$, we can show that the minimum-weight outgoing edge of F can be found within the time bound (i.e., $t_F + T_{k-1} - 2^{k+1}$ as stated in the hypothesis).

First, we shall prove by contradiction that all the subfragments of F are of size less than 2^k . Assume that there exists a subfragment F_i of F with size $2^k \leq SZ_i < 2^{k+1}$. F_i will either merge into and obtain the level of another higher level fragment F_j , or merge with F_j over the same outgoing edge. For the former case, since $SZ_i \geq 2^k$, $SZ_j \geq 2^k$ and $F \supseteq F_i \cup F_j$, we have $SZ \geq SZ_i + SZ_j \geq 2^{k+1}$, which leads to a contradiction. For the latter case, it is required that we show that $t_c < t_{F_i} + T_k + 2^{k+1}$, where t_c is the time when a *CONNECT* message is sent from F_j to F_i . From our algorithm, the coordinator(s) of F_i will only broadcast the *INITIATE*(F_i, k, S, t_{B_i}) message at time t_{B_i} , where $t_{B_i} > t_{F_i} + T_k + 2^{k+1}$; thus we have $t_{B_i} > t_c$ if the inequality relation for t_c is true. This would imply that F_j has merged with F_i before F_i sends a *TEST*(F_i) message to F_j ; this would then lead to a contradiction that F_i and F_j are merged together over the same outgoing edge. The remaining paragraph will prove the inequality relation for t_c . Since $SZ_j \leq SZ - SZ_i < 2^k$, F_j will find its minimum-weight outgoing edge before time $t_{F_j} + T_k - 2^k$, from the induction hypothesis, and will take less than 2^k time units to send a *CONNECT* message to F_i ; thus, $t_c < t_{F_j} + T_k$. As the distance between the coordinator of F_i and any nodes in F is less than 2^{k+1} , we have $t_{F_j} < t_{F_i} + 2^{k+1}$ and $t_c < t_{F_j} + T_k < t_{F_i} + T_k + 2^{k+1}$.

Using the argument in the previous paragraph, since all subfragments F_j of F are of sizes less than 2^k , t_c , the time when the *CONNECT* message is sent out from each F_j , will be strictly less than $t_{F_j} + T_k$. Since $t_{F_j} < t_F + 2^{k+1}$, we have $t_c < t_F + T_k + 2^{k+1}$, which in term is also less than F 's broadcast time of the *INITIATE*(F, k, S, t_B) messages, t_B . Thus, all nodes in F would have received their corresponding fragment identity no later than $t_B + 2^{k+1}$. As from the algorithm, all the *TEST* messages are only sent after time $t_B + 2^{k+1}$, no intrafragment edge will be accepted and the minimum-weight outgoing edge can be correctly found.

Now, let us find an upper bound for the time when F finds its minimum-weight outgoing edge. Let k_0 be the level of F when it is initially formed. Without loss of generality, let us assume that before F determines its minimum-weight outgoing edge, there is a series of level updates from k_0 to k_1 , to k_2 , \dots , and finally to k . Because there are at most 2^{k_i+1} *INITIATE*, *TEST*, *REJECT/ACCEPT*, *REPORT* messages for the i th level update and all the messages are transmitted without delay, each level update will not be broadcast after $t_F + T_k + 2^{k+1}$, the time when all subfragments would have been merged together, the total time for F to find its minimum-weight outgoing edge is less than

$$\begin{aligned} & t_F + T_k + 2^{k+1} + 4 \cdot (2^{k_1+1} + 2^{k_2+1} + \dots + 2^{k+1} + 2^{k+1}) \\ & < t_F + T_k + 2^{k+1} + 4 \cdot (2^1 + 2^2 + \dots + 2^{k+1} + 2^{k+1}) < t_F + T_k + 26 \cdot 2^k \\ & = t_F + T_{k+1} - 2^{k+1}. \quad \square \end{aligned}$$

THEOREM 2. *The time complexity of our algorithm is $55n$.*

Proof. From Lemma 4, the algorithm terminates by time $t = t_F + T_{\log n+1} - 2^{\log n+1}$. Since $t_F < n$ and $T_{\log n+1} < 56n$, we have $t < 55n$. \square

The message complexity of our algorithm is $4n \log n + 4e$ rather than $5n \log n + 4e$ (including the initial $2e$ *AWAKE* messages). This is because *ACCEPT* messages are not needed in our algorithm; the reception of a *TEST* message can be assumed by default if no message is reported within two time units after it is sent. From Theorem

2, at most $\log n + 6$ bits are needed to encode the broadcast time. The fragment level can be encoded in $\log \log n$ bits and fragment size in $\log n$ bits. As three bits are enough to distinguish different types of messages, each message contains at most one edge weight or one node identity plus $(\log n + \log \log n + 9)$ bits.

4. Conclusion. We have improved the time complexity of the asynchronous and synchronous distributed algorithms for the minimum-weight spanning tree problem. Since our algorithms work for any arbitrary graph, it is obvious that $\theta(n)$ time complexity is needed for the problem. Our synchronous distributed algorithm for finding the minimum-weight spanning tree is not only message-optimal but also time-optimal. Even though Awerbuch has shown that $\theta(n)$ time is also achievable in the asynchronous network for this problem, it is not surprising to notice that a synchronous algorithm for a problem can always outperform its asynchronous counterpart [2], [9]–[11]. One of the open problems is to determine what extra information a synchronous algorithm has over an asynchronous one.

We have improved the time complexity of the Gallager, Humblet, and Spira's asynchronous algorithm for finding the MST of a graph to $O(n \log^* n)$. Although the worst-case message complexity remains $O(e + n \log n)$, our algorithms seem to have a larger message complexity because *REPORT* messages must be sent whenever a new fragment is formed. On the other hand, we can argue that our algorithms may use fewer messages on the average because whenever a large fragment is formed, its level will be increased sufficiently at the earliest possible instance and this may eliminate a number of unnecessary messages. It would be interesting to conduct some simulations to study the complexities of our algorithms when compared with Gallager's.

Acknowledgments. The authors thank M. Y. Chan for her careful reading and for pointing out a mistake in our synchronous algorithm. The authors are indebted to the anonymous referees who have given various valuable comments in improving the readability of the paper, in particular, the simplified example with time complexity $\Omega(n \log^* n)$ for the asynchronous algorithm.

REFERENCES

- [1] B. AWERBUCH, *Hierarchical routing with small memory per node*, unpublished manuscript, October 1985.
- [2] ———, *Complexity of network synchronization*, J. Assoc. Comput. Mach., 32 (1985), pp. 804–823.
- [3] B. AWERBUCH AND R. G. GALLAGER, *Distributed breadth-first-search algorithms*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Sciences, IEEE Computer Society, Washington, DC, 1985, pp. 250–256.
- [4] B. AWERBUCH AND S. MICALI, *Dynamic deadlock resolution protocols*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1986.
- [5] B. AWERBUCH, *Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems*, in Proc. 19th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1987, pp. 230–240.
- [6] ———, *Linear time algorithm for minimum network partition*, TR Memo MIT/LCS/TM-350, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, March 1988.
- [7] O. BORUVKA, *O jistém problému minimálním*, Práce Mor. Přírodověd. Spol. v Brně (Acta Societ. Scient. Natur. Moravicae), 3 (1926), pp. 37–58.
- [8] ———, *Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí*, Elektrotechnický obzor, 15 (1926), pp. 153–154.
- [9] M. Y. CHAN, *Election and symmetry breaking in synchronous general networks*, TR-B7-86, University of Hong Kong, Hong Kong, July 1986.
- [10] E. GAFNI AND Y. AFEK, *Time and message bounds for election in synchronous and asynchronous complete networks*, in Proc. 4th Annual ACM Symposium on Principles of Distributed Computing, Association for Computing Machinery, New York, 1985, pp. 186–195.
- [11] E. GAFNI, *Improvements in the time complexity of two message-optimal election algorithms*, in Proc. 4th Annual ACM Symposium on Principles of Distributed Computing, Association for Computing Machinery, New York, 1985, pp. 175–185.

- [12] R. G. GALLAGER, *Finding a leader in a network with $O(E + N \log N)$ messages*, Massachusetts Institute of Technology, Cambridge, MA, 1977.
- [13] R. G. GALLAGER, P. A. HUMBLET, AND P. M. SPIRA, *A distributed algorithm for minimum-weight spanning trees*, ACM Trans. Programming Languages and Systems, 5 (1983), pp. 66–77.
- [14] R. L. GRAHAM AND P. HELL, *On the history of the minimum spanning tree problem*, Ann. Hist. Comput., 7 (1985), pp. 43–57.
- [15] P. A. HUMBLET, *A distributed algorithm for minimum weight directed spanning trees*, IEEE Trans. Comm., 31 (1983), pp. 756–762.
- [16] J. PACHL, E. KORACH, AND D. ROTEM, *Lower bounds for distributed maximum-finding algorithms*, J. Assoc. Comput. Mach., 31 (1984), pp. 905–918.
- [17] D. PELEG AND E. UPFAL, *A tradeoff between size and efficiency for routing tables*, in Proc. 20th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1988, pp. 43–52.
- [18] P. M. SPIRA, *Communication complexity of distributed minimum spanning tree algorithms*, in Proc. 2nd Berkeley Conference on Distributed Data Management Computing Networks, Berkeley, CA, June 1977.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.