DISTRIBUTED ALGORITHMS FOR COMPUTING ALL INTERNODE DISTANCES AND FOR CONSTRUCTING A MINIMUM-WEIGHTS SPANNING TREE IN A FULLY CONNECTED BROADCAST NETWORK

BY

Yaron I. Gold* and Shlomo Moran**

Technical Report #297
November 1983

* Electrical Engineering and Computer Science, University of Connecticut,
** Computer Science Department
Technion - Israel Institute of Technology, Haifa, Israel
DISTRIBUTED ALGORITHMS FOR COMPUTING ALL INTERNODE DISTANCES AND FOR CONSTRUCTING A MINIMUM-WEIGHT SPANNING TREE IN A FULLY CONNECTED BROADCAST NETWORK

by

Yaron I. Gold
Electrical Engineering and Computer Science
University of Connecticut

and

Shlomo Moran
Computer Science Department
Technion, Israel Institute of Technology

ABSTRACT

Two distributed algorithms are presented for a network using a common communication channel (e.g. radio) in which all nodes are within signal range and in line of sight of each other: (a) an algorithm to compute all \( \binom{N}{2} \) internode distances (in terms of propagation delays) in the network. The algorithm requires only 2 messages per node, and provides each node with the distances to all other nodes. (b) An algorithm for constructing a minimum-weight spanning tree (MST) in such a network. This algorithm starts out with the information provided by (a) and ends with each node possessing the explicit knowledge of the full MST. The algorithm requires at most \( \log_2 N \) messages per node. The internal processing in each node needs \( O(N \log N) \) time and \( O(N) \) space. All messages required by (a) and (b) contain at most one edge weight plus \( 2 \log_2 N \) bits. Some possible applications of the algorithms are: position-location [16], tuning acknowledgement time-out mechanisms, tuning the scheduling functions of access protocols that are sensitive to individual internode propagation delays, and selecting performance effective transmission sequences for round-robin access protocols.
1. **INTRODUCTION**

Multi-access broadcast channels (e.g. radio) have become popular communication media for computer networks and other distributed systems [5]. Networks using such a shared channel differ from networks that use point-to-point links (e.g. a telephone network) in several ways. Two major distinctive features of a multi-access broadcast channel are:

(a) Each single transmission is received by more than one node (and often by all nodes) and is hence considered a broadcast.

(b) All nodes can access the channel for transmission at any time, but a node can receive correctly only one transmission at a time. Simultaneous or overlapping transmissions cause loss of information.

Due to (b) the nodes must use some mechanism that controls the access to the common channel. Such a mechanism is called an "access protocol" and a variety of such protocols exists [15]. An access protocol that completely avoids overlapping transmissions is said to be "conflict-free". A family of distributed conflict-free access protocols, called "round-robin token-passing" protocols make very efficient use of the channel. These protocols schedule the nodes to access the channel in a cyclic sequence. A node that does not want to use its access rights (i.e., has nothing to transmit) "passes the token" to the next node in the sequence. The token may be an actual message or may be implemented as a distributed scheduling function using "time-out" mechanisms (a virtual token) [1, 8, 9, 13]. In the most efficient token-passing protocols [e.g. 8, 9] the time it takes to pass the token form some node, say, node-i, to the next node in the sequence, say, node-j, is equal to the propagation delay \(d(i,j)\), between these two nodes. It can be shown [8] that the performance of such a protocol depends largely on the time it takes to pass the token once around all the nodes in the network. An optimal token-passing sequence is one that minimizes this time. If the network is represented as a weighted graph with the nodes as vertices and the propagation delays as the costs of the edges (i.e. \(d(i,j)\) is the cost of edge \((i,j)\)), then selecting an optimal sequence is equivalent to solving the (Euclidean) Traveling Salesman Problem (TSP).
with respect to this graph, a problem that is NP-hard. In a network with constant topology an optimal sequence can be computed once before network initialization and used thereafter. However if the topology is dynamic (e.g., the network contains mobile nodes) it may be necessary to recompute the transmission sequence from time to time to maintain good performance. Due to the difficulty of the TSP, finding an optimal sequence may be prohibitively costly (the nodes may have to spend most of their time updating the sequence) and the result outdated (the topology may have changed considerably by the time the update is completed). Therefore a fast approximation algorithm should be used to find a suboptimal sequence. Several algorithms for constructing a suboptimal TSP tour involve a minimal weight spanning tree (MST) [2,7,11]. Distributed algorithms for constructing an MST in a network with point-to-point links (e.g., a telephone network) are described in [3,6,14]. These algorithms are either unsuitable for a multi-access channel or loose their efficiency when adapted for use with such a channel. An efficient distributed algorithm specially designed for constructing an MST in a multi-access broadcast network is therefore desirable. When edge costs are defined as internode propagation delays an efficient algorithm for acquiring this information is also needed, especially in networks with dynamic topologies.

For our discussion we assume a model with the following properties:

(a) The network consists of \( N \) nodes, each with internal processing capability and with the ability to transmit and receive messages.

(b) The nodes share a common communication channel which enables broadcasting (more than one receiver for each transmission) but does not allow correct reception of simultaneous or overlapping transmissions.

(c) The nodes are all "in line-of-sight" and within signal range of each other, so the network can be represented as a fully-connected graph, \( G \). Each transmission is therefore received by each node after a delay equal to the propagation delay between the transmitting node and the receiving node.
A cost \( d(i,j)=d(j,i) \) is associated with each edge \((i,j)\). This cost is equal to the propagation delay between node-\(i\) and node-\(j\) \((G\) is thus a weighted, undirected graph). It is also assumed that the costs satisfy the triangle inequality, and that the nodes have no a priori knowledge of these costs.

(e) A cyclic order is imposed on the nodes at any given time. Each node knows the node immediately preceding it in that order. Without loss of generality we can assume that node indexes are assigned according to this order \((i.e., \text{node-}1\text{ immediately precedes node-}\,(i+1), \text{ and node-}N\text{ precedes node-}1\). In the following, "+" or "-" in the context of node indexes will always be modulo-\(N\). A natural access protocol for such a network is a round-robin protocol that has this order as its transmission order.

(f) The amount of topological change that may take place in the network during a time period similar to the average internode propagation delay is negligible, i.e., the network topology changes very slowly relative to signal propagation speed (normally around the speed of light).

In Section 2 we describe a distributed protocol by which the nodes can measure the topological information needed for constructing an MST of the network. As a result of executing this protocol the network, as a whole, knows all \(\binom{N}{2}\) internode propagation delays. However, this information is distributed among the \(N\) nodes - each node knows the propagation delays from itself to all other nodes in the network. The protocol takes advantage of the broadcasting capability of the channel to perform the task with only two short messages transmitted by each node. The number of bits in one message is \(\log_2 N\) and in the other \(\log_2 N + p\), with \(p\) the number of bits necessary to represent one internode propagation delay. The total number of messages, \(2N\), is much less than the \(O(N^2)\) messages required in a point-to-point network or if each node were to obtain the same information independently.

In Section 3 we present a distributed algorithm for constructing an MST of a network that corresponds to the model described above. This algorithm uses the information produced by the protocol of Section 2, i.e., each node initially knows the costs of the edges from itself to all other nodes. After execution each node
has explicit knowledge of the full MST. The algorithm requires at most \( \log_2 N \) cycles, each cycle consists of at most one message transmitted by each node. The total number of messages is therefore \( N \log_2 N \) at the most. Each message is at most \( 2 \log_2 N + p \) bits long. Each cycle also requires some internal processing that is performed in parallel by all nodes. This processing has a total time complexity of \( O(N \log N) \) per node, and space complexity of \( O(N) \) per node. (Internal processing time is normally assumed much shorter than communication time).

A summary and a conclusion are given in Section 4 in which we also discuss the construction of a suboptimal token-passing sequence, using the MST produced by the algorithm of Section 3.

2. MEASURING INTERNODE PROPAGATION DELAYS

The distributed protocol described below is executed by the nodes in a multi-access broadcast network in order to obtain all internode propagation delays, \( d(i,j) \), needed to construct the MST by the algorithm in Section 3. After executing this protocol each node knows the propagation delays between itself and all other nodes. The communication part of the protocol consists of two cycles and requires each node to broadcast 2 short messages, one in each cycle. All messages are received by all nodes, though the arrival pattern (the intervals between the arriving messages) differs from one node to another.

The first cycle is initiated by some node, say node-1, which broadcasts a short message called a "trigger". Each node-1 broadcasts a trigger as soon as it has received the trailing edge (end) of the trigger broadcast by node-(i-1) (node-(i-1) is the node that immediately precedes node-1 in the round-robin transmission order). When the last node, node-N which also precedes node-1, has broadcast and the trailing edge of its trigger was received by node-1, node-1 concludes the first cycle by transmitting one more trigger. The triggers are so named since they control (start and stop) a high-speed counter that each node uses to measure certain time intervals, as explained below, and also since the arrival of the trailing edge of node-1's trigger at node-(i+1) invokes the immediate transmission of the latter node's trigger. Each trigger contains the index of its source, and hence consists of \( \log_2 N \) bits. Figure 1 shows the arrival pattern for node-3 in a 5 node example network.
During the first cycle the nodes measure certain time intervals between arriving triggers: let $I_i(m, n)$ be the duration of the time interval between the trailing edge (end) of node-$m$'s trigger and the subsequent leading edge (start) of node-$n$'s trigger, as seen by node-$i$. Node-$i$ measures the $N-1$ intervals $I_i(m, m+1)$ for all $m=1$ (see Figure 1). $I_i(i-1, i)$ is always (by definition) zero and need not be measured. The concluding trigger by node-$1$ is used to measure $I_i(N, 1)$. Since these intervals are non-overlapping, a single high-speed counter which is started and stopped by the appropriate trigger edges is sufficient for all $N-1$ measurements. It is easy to show (see Figure 2) that:

$$(1) \quad I_i(m, m+1) = d(m, m+1) + d(m+1, i) - d(m, i).$$

In particular $I_i(i, i+1) = 2d(i, i+1)$, so at the end of the first cycle each node-$i$ knows the values $I_i(m, m+1)$ for all $1 \leq m \leq N$ and the value of $d(i, i+1)$.

During the second cycle each node-$i$ broadcasts (in its turn) a message containing its index, $i$, and also the value of $d(i, i+1)$. These messages consist of $\log_2 N + p$ bits, with $p$ the number of bits needed to represent one internode propagation delay. Node-$1$ can, in fact, transmit this information in its second trigger which concluded the first cycle, and need not transmit a third message. Thus, at the end of the second cycle each node-$i$ knows the values of $d(m, m+1)$ and $I_i(m, m+1)$.
for all $1 \leq m \leq N$.

Next we show that from these values node-1 can compute $d(j, i)$ for all $1 \leq j \leq N$, $j \neq i-1$ (the value $d(i-1, i)$ is known directly from the message by node-$(i-1)$).

**Theorem 1.** For $j \neq i-1$ $d(j, i)$ is given by:

$$d(j, i) = \sum_{m=j}^{i-1} [d(m, m+1) - I_1(m, m+1)].$$

**Proof:** From (1) we have that the summand in the right hand side of (2) can be expressed by:

$$d(m; m+1) - I_1(m, m+1) =$$
$$= d(m, m+1) - [d(m, m+1) + d(m+1, i) - d(m, i)]$$
$$= d(m, i) - d(m+1, i)$$

so to prove the theorem we may prove that for $j \neq i-1$ $d(j, i)$ is given by:

$$d(j, i) = \sum_{m=j}^{i-1} [d(m, i) - d(m+1, i)].$$

All terms in the summation of the right hand side of (3) cancel, except $d(j, i)$ from the first summand and $-d(i, i)$ from the last summand. Since by definition the latter equals zero the theorem is proven. []

Note that if the network uses SOSAM [9] as its round robin access protocol, and uses the protocol described in [10] to update SOSAM's scheduling function, then the additional overhead required for the protocol described here is minimal, as its communication part is similar to that of the update protocol in [10].

3. **CONSTRUCTING THE MINIMAL SPANNING TREE**

At this point each node knows the distances (propagation delays) from itself to all other nodes. In this section we provide a distributed algorithm that uses this
information to construct a minimal spanning tree (MST) of the network. This MST can then be used to construct a suboptimal Traveling Salesman tour to serve, for instance, as the new cyclic transmission order in the network.

3.1 Outline of the Algorithm

Using the regular round robin transmission sequence and access-protocol the nodes exchange information and process it in one or more cycles. In each cycle the number of connected components of the partially constructed MST is decreased by a factor of at least 2. This is done by joining certain components with the shortest edge that connects them. To do this, each node transmits (in its turn) a message containing the identity and length of the edge between itself and the nearest node in a different component (to reduce the number of messages, if a message containing a shorter edge has already been received from another node in the same component, then the current node, knowing its edge has no chance to be the shortest, "passes the token" to the next node in the sequence and does not transmit a message). All nodes keep track of the information on the "forest" constructed so far and on the minimal edges received during a given cycle. At the end of the cycle each node determines the minimal outgoing edge from each component and uses this information to construct the new, larger components (a minimal outgoing edge from a component is the shortest edge that has exactly one end in that component). Components are joined into larger ones by adding their minimal outgoing edges to the "forest" (the partially constructed MST). The resulting combined component are used as input to the subsequent cycle.

The MST is completed in at most $\log_2 N$ cycles. The number of messages transmitted in each cycle is at most $N$, so the total number of messages needed to construct the MST is $N \log_2 N$ at most.

A cycle consists of both communication and internal processing. Most of the internal processing time is spent at the end of the cycle on joining components into larger ones. It is assumed that internal processing time is negligible relative to communication time. However, even if this assumption is false the total complexity of the algorithm (see Section 3.4) is still $O(N \log N)$, since the time complexity per cycle of the internal processing is also $O(N)$. The algorithm employs a data structure with space complexity $O(N)$ per node. At the end of the algorithm each node
explicitly knows the full structure of the MST, i.e., to become a member of the MST.

3.2 The Data Structure

3.3 During execution of the algorithm each node uses the information received from messages broadcast by other nodes to manipulate its own local data structures. These data structures, similar in all nodes, consist of the following (for an arbitrary node j):

(a) An array of records, NEIGHBOR[1..N]. Each record, NEIGHBOR[j], consists of two fields: NEIGHBOR[j].x, which is a node index (each record contains a distinct x-index and all indexes occur except x is node index), and NEIGHBOR[j].y, which is the edge length of the outgoing edge from the corresponding component. During each cycle the

(b) An adjacency list representing the fully or partially constructed MST.

(c) An array of pointers, NEXTEL[1..N], which is used to link all nodes that are elements in the same component to form a circular list. The circular linked list allows fast sequential access to elements of a given component and facilitates "joining" of components. In each such list one element is identified as the "representative element" of the corresponding component.

(d) An array REPELEMENT[j] points to (contains the index of) the representative element of the component containing node j.

(e) Two arrays, NEXT[1..N] and PREV[1..N] which are used to link the components via their representative elements in a doubly linked list. The contents of NEXT[j] or of PREV[j] is meaningful only if j is an index of a representative element. COMPLIST is a pointer to the first representative element in this list.

(f) An array of records EDGE[1..N]. Each record, EDGE[j], has three fields: EDGE[j].n1 and EDGE[j].n2 which are node indices and EDGE[j].Length = d(n1, n2). The contents of EDGE[j] is meaningful only if j is the index of a representative element, so only one record per component is actually used, and it contains the end nodes and the length (in terms of propagation delay) of the minimal outgoing edge from the corresponding component. During each cycle the
information in these records is updated, and then used to join components.

3.3 The Algorithm

The algorithm outlined in Section 3.1 is provided in this section in PASCAL-like notation. The algorithm consists of a main procedure called CONSTRUCT_MST, which calls several other procedures, the most important of which is JOIN_COMPONENTS. The variable $i$ that occurs in the procedures is the index of the node executing the algorithm. To illustrate the operation of CONSTRUCT_MST we consider an example network with $N=19$ nodes. Figure 3 provides three partial "snap-shots" of the data structure of a node in this network. Each snap-shot corresponds to a different cycle and shows the status of the data structure just before executing JOIN_COMPONENTS. The solid lines between the circles represent the MST edges that are already in the adjacency list. The dotted lines represent the edges just about to be inserted, based on the information in $\text{EDGE}[j].n_1$ and $\text{EDGE}[j].n_2$ (also shown in the figure).

---

**Figure 3** - Three partial snap-shots of
the data structure for a 19 node example network.
(a) before first execution of JOIN_COMPONENTS
(b) before second execution
(c) before third and last execution.
---
procedure CONSTRUCT_MST

INITIALIZE
white number of components > 1 do
for k := 1 to N do
if k = i then BROADCAST (message {d, i, j})
else RECEIVE (message {d, k, j})
JOIN_COMPONENTS
end /* of while loop */
end /* of CONSTRUCT_MST */

procedure INITIALIZE

SORT (NEIGHBOR) /* by increasing order of d */
NEAREST := 1
initialize adjacency list
COMPLIST := 1
for j := 1 to N do
if j > 1 then PREV [j] := j-1
if j < N then NEXT [j] := j+1
EDGE [j].Length := \infty
end /* of for loop */
end /* of INITIALIZE */

procedure BROADCAST (message {d, i, j})

/* set d equal to the distance to the nearest node in a different component and j equal to the index of that node */

d := NEIGHBOR [NEAREST].d
j := NEIGHBOR [NEAREST].n
REP := REP_ELEMNT [1]
if d < EDGE [REP].Length
then
/* update EDGE [REP] */
with EDGE [REP] do
Length := d
n_1 := i
n_2 := j
TRANSMIT (message {d, f, j})
else
TRANSMIT (message {nil, i, nil}) /* token only, may be omitted in virtual-token systems */
end /* of BROADCAST */
procedure RECEIVE (message \( \{d, k, j\} \))

\[
\text{if } d \neq \text{nil} \text{ then}
\]
\[
\quad \text{REP} := \text{REP_ELEMNT} \[k\]
\]
\[
\quad \text{with } \text{EDGE} \[\text{REP}\] \text{ do}
\]
\[
\quad \text{Length} := d
\]
\[
\quad n_1 := k
\]
\[
\quad n_2 := j
\]
\[
\text{end} /* \text{or if */}
\]
\[
\text{end} /* \text{of RECEIVE */}
\]

procedure JOIN_COMPONENTS

\[
\text{COMP} := \text{COMPLIST}
\]
\[
\text{while } \text{COMP} \neq \text{nil} \text{ do}
\]
\[
/* \text{set JOIN to the index of the nearest node in the nearest component */}
\]
\[
\text{JOIN} := \text{EDGE} \[\text{COMP}\].n_2
\]
\[
/* \text{if the components have not been joined yet, join them: */}
\]
\[
\text{if } \text{REP_ELEMNT} \[\text{JOIN}\] \neq \text{REP_ELEMNT} \[\text{COMP}\] \text{ then}
\]
\[
/* \text{initialize walk through the elements of the component */}
\]
\[
\text{ELEMENT} := \text{COMP}
\]
\[
/* \text{walk through component and update representative element */}
\]
\[
\text{repeat}
\]
\[
\quad \text{REP_ELEMNT} \[\text{ELEMENT}\] := \text{REP_ELEMNT} \[\text{JOIN}\]
\]
\[
\quad \text{ELEMENT} := \text{NEXT} \[\text{ELEMENT}\]
\]
\[
\text{until } \text{ELEMENT} = \text{COMP}
\]
\[
/* \text{delete component from component list */}
\]
\[
\text{if } \text{PREV} \[\text{COMP}\] \neq \text{nil} \text{ then}
\]
\[
\quad \text{NEXT} \[\text{PREV} \[\text{COMP}\]\] := \text{NEXT} \[\text{COMP}\]
\]
\[
\text{else } \text{COMPLIST} := \text{NEXT} \[\text{COMP}\]
\]
\[
\text{if } \text{NEXT} \[\text{COMP}\] \neq \text{nil} \text{ then}
\]
\[
\quad \text{PREV} \[\text{NEXT} \[\text{COMP}\]\] := \text{PREV} \[\text{COMP}\]
\]
\[
/* \text{join the element lists of both components */}
\]
3.4 Proofs of Correctness and Complexity

A formal proof of the correctness and complexity of `CONSTRUCT_MST` is given below. Like in [6] we first consider the case where all edge lengths (costs) are distinct, and then extend the definition of edge costs to impose this property on arbitrary graphs with distinct node identities.

Let $G = (V, E)$ be a weighted, undirected graph, with $w(e)$ the cost of edge $e$, and let $U$ be a subset of $V$. An edge is said to be "outgoing" from $U$ if it has exactly one end node in $U$. An edge $e$ is a "minimal outgoing edge" if there exists $U$, a proper subset of $V$, such that $e$ is outgoing from $U$, and for every edge $e'$ outgoing from $U$, $w(e) \leq w(e')$. If $e$ is the only minimal outgoing edge from some $U$ then $e$ is a "unique minimal outgoing edge". The property of minimal spanning trees which implies the correctness of our algorithm is given by:

**Lemma 1**: If $e$ is a unique minimal outgoing edge then $e$ is an edge in every MST of $G$.

**Proof** (by contradiction): Let $e$ be a unique minimal outgoing edge (from some subset of vertices, $U$) and assume that there is an MST of $G$, say $T'$, which does not contain $e$. Then by adding $e$ to $T'$ one circuit is created, all whose edges except $e$ are in $T'$. At least one of these edges, say $e'$, is also outgoing from $U$. 
By the uniqueness of $e$, $w(e') > w(e)$. Thus by replacing $e'$ by $e$ we obtain a spanning tree $T$ whose weight is smaller than that of $T'$ — a contradiction.

**Corollary:** If all edge costs are distinct then every minimal outgoing edge is in every MST of $G$.

**Proof:** If all edge costs are distinct then every minimal outgoing edge is a unique minimal outgoing edge, and the corollary follows from Lemma 1. [1]

**Theorem 2:** If all edge costs in $G$ are distinct then CONSTRUCT_MST halts after at most $\log_2 N$ cycles, with the adjacency lists in the local memory of each node containing the edges of the (unique) MST of $G$.

**Proof:** Simple induction shows that at the end of the $i$-th cycle the number of connected components constructed so far is at most $N/2^i$. Since the algorithm halts when there is a single component (which, of course, is spanning the whole graph $G$), it halts after at most $\log_2 N$ cycles. Furthermore, all the edges in the spanning subgraph constructed by the algorithm are minimal outgoing edges (this follows directly from the manner of construction). This implies, by the distinctness of the weights and the corollary above, that all these edges are in every MST of $G$. Thus the constructed spanning subgraph is included in every MST of $G$, and hence it must be the (unique) MST of $G$. [1]

The proof above requires that edge costs be distinct. If some edges are equal in length then it is necessary to impose an order on them, for instance by using the readily available identities of their end nodes (see e.g. [6]).

**Theorem 3:** The internal processing required by CONSTRUCT_MST in each node has $O(N)$ space complexity and $O(N \log N)$ time complexity.

**Proof:** Each node uses one adjacency list and five arrays, each requiring $O(N)$ space.

The time needed by INITIALIZE is $O(N \log N)$ for sorting the $N-1$ edges in NEIGHBOR and $O(N)$ for initializing the adjacency list and the execution of the for loop. BROADCAST requires $O(1)$ time and is executed once per cycle, and
RECEIVE requires $O(1)$ time and is executed $N-1$ times per cycle, so the total time required by the for loop in CONSTRUCT_MST is $O(N)$ per cycle. JOIN_COMPONENTS requires $O(N)$ time per cycle for updating and joining components (the first while loop) and a total of $O(N)$ time for advancing NEAREST (the second while loop). Since there are at most $\log_2 N$ cycles, the total internal time complexity is $O(N\log N)$ for each node.

4. CONCLUSION

In this paper we described two distributed algorithms for fully connected networks that use a multi-access and broadcast channel as their communication medium. The first is a protocol that provides each node with the distances (in terms of propagation delays) to all other nodes. This information is required by the second algorithm in order to construct a minimum-weight spanning-tree (MST), however it may have other applications, such as position-location [16], tuning acknowledgement time-out mechanisms or tuning the scheduling functions of access protocols such as SOSAM [9,10] and HYPERchannel\textsuperscript{TM} [1a] which are sensitive to individual internode propagation delays. The second algorithm uses the information obtained by the first in order to construct an MST. Both algorithms take advantage of the broadcast property of the channel to reduce the number of messages that each node must transmit during the execution. The algorithm for measuring internode distances requires $2N$ messages (2 per node) - much less than the $O(N^2)$ messages that are required for the same task in a point to point network, or if each node were to obtain the same information independently. The algorithm for constructing the MST requires $O(N\log N)$ messages, in contrast with the $O(N^2)$ messages that are required for constructing the MST in a fully connected point to point network [12] (the best known distributed algorithms for constructing an MST in a point to point networks require $O(N\log N + e)$ messages, where $e$ is the number of edges in the network graph [6]).

Both algorithms interact well with round-robin access protocols such as SOSAM, due to the inherent cyclic order in which messages are broadcast, namely, one message per node per cycle. Note that in multi-access networks the number of required messages is not necessarily a true indicator of protocol efficiency, since inefficient channel access control may considerably increase the total time required for execution. This is not the case when the network avoids unnecessary delays between
messages by using an efficient access protocol. The total time to execute the algorithm is then roughly equal to message transmission time. Furthermore, the "one message per node per cycle" nature of the transmissions helps distribute the communication load evenly among the nodes of the network.

It is interesting to note that the time and space complexities of the internal processing required by CONSTRUCT_MST (i.e. \(O(N)\) space and \(O(N \log N)\) time per node, or \(O(N^2)\) space and \(O(N^2 \log N)\) time total) are the same (space) or very close (time) to the complexities of the best known sequential MST algorithms for fully connected graphs (i.e. \(O(N^2)\) both time and space [4]).

After execution of CONSTRUCT_MST each node possesses explicit knowledge of the full MST. This fact minimizes (or even eliminates) the need for further communication among nodes when the MST info is required for further processing, such as constructing an approximate TSP tour. As mentioned in the introduction, this paper was motivated by the need for dynamic selection of a round-robin transmission sequence. This selection is equivalent to constructing an optimal (or suboptimal) TSP tour. A variety of approximation algorithms for this problem are known, some of which use a minimal spanning tree [2,7,11]. One such algorithm, which seems to be suitable for our application performs a Depth First Search (DFS) on the MST to construct a tour that is at most twice as long as the optimal tour (see e.g. [7]). This algorithm has \(O(N)\) time complexity and requires knowledge of the MST edges only. It can therefore be used in conjunction with the algorithm CONSTRUCT_MST given in Section 3 which provides each node with an explicit representation of the full MST. (An extension of this algorithm, due to Christofides [2], guarantees a relative error of at most 1.5, but has a time complexity of \(O(N^3)\) and may require the knowledge of the weights of many edges that are not in the MST).

To construct the approximate TSP tour the algorithm in [7] may be executed independently by all nodes. However, to assure that all nodes produce the same sequence, a unique root must be agreed upon (say, node 1). Additionally, The nodes must select branches in a unique order during the DFS. The order of appearance in the MST adjacency lists, which is the same in all nodes, can be used for this purpose.

Finally, the amount of computation required for selecting a token-passing
sequence can be reduced somewhat by noting that for this purpose each node must know only the index of the node immediately preceding it in the sequence. A node may, therefore, stop constructing the tour as soon as it has first encountered its own index in the partially constructed tour.
REFERENCES


Figure 1 - Trigger arrival pattern for node-3 in a 5 node example network.

\[ I_3(m,m+1) = d(m,m+1) + d(m+1,i) - d(m,i) \]

Figure 2 - \( I_3(m,m+1) \) expressed as a function of internode propagation delays.
Figure 3 - Three partial snapshots of the data structure for a 19 node example network.

(a) before first execution of JOIN_COMPONENTS
Figure 3 (cont.) - (b) before second execution
Figure 3 (cont.) - (c) before third and last execution.