A Distributed Algorithm for Constructing Minimal Spanning Trees in Computer-Communication Networks

by

Yogen K. Dalal

June 1976

Technical Report No. 111

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76C-0093 and the National Science Foundation under research grant MCS73-07973-Al, 2.

A Distributed Algorithm for Constructing Minimal Spanning Trees in Computer-Communication Networks

Technical Report No. 111

June, 1976

Yogen K. Dalal
Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California

ABSTRACT

This paper presents a distributed algorithm for constructing minimal spanning trees in computer-communication networks. The algorithm can be executed concurrently and asynchronously by the different computers of the network. This algorithm is also suitable for constructing minimal spanning trees using a multiprocessor computer system. There are many reasons for constructing minimal spanning trees in computer-communication networks since minimal spanning tree routing is useful in distributed operating systems for performing broadcast, in adaptive routing algorithms for transmitting delay estimates, and in other networks like the Packet Radio Network.

Key Words and Phrases

Minimal Spanning Trees, Graphs, Distributed Control, Multiprocessing, Computer-Communication Networks, Operating Systems, Distributed Computing, Routing

CR Categories 3.81, 4.32, 4.35, 5.32

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76C-0093 and the National Science Foundation under research grant MCS73-07973-A1,2.

A Distributed MST Algorithm

1. Introduction

Distributed operating systems that support a distributed computing environment [Farber72a, Thomas73, Crocker75] may often have to make available to a user process a remotely resident resource. The resource may be capable of migration (e.g. files in a distributed file system or processes capable of performing specialized functions), or could be the least expensive copy of a duplicated resource [Cosell75, Dalal76]. In order to find such a resource the requesting host may have to send a request message to all hosts potentially capable of supplying the resource. In general, this set of hosts will be a subset of all the hosts in the network. For the purpose of this paper, however, we consider the problem of delivering the message to all hosts. The requestor will be said to broadcast the message (to all hosts).

The efficiency of the broadcast is greatly dependent on the nature of the particular subnet over which it is attempted. The structure of the subnet also influences the design of the broadcast protocol chosen to find resources. For example, multiaccess channels, like those available in the ALOHA system [Abramson70], the Ethernet [Metcalfe75], satellite networks [Abramson73], or ring networks [Farber72] lend themselves very well to broadcast protocols since the very nature of the subnet makes every transmission available to all hosts. Circuit Switched Networks (CSN) provide point-to-point communication, and so broadcast is done either by having a separate circuit between the broadcaster and each receiver, or by creating a multidrop circuit, that behaves like a ring, between the broadcaster and the receivers. Packet Switched Networks (PSN) have storage and a (small) holding time at every switching node, and so can be thought of as providing statistical time division multiplexed communication. PSNs are more suitable for performing broadcast than CSNs, as advantage can be taken of the packet mode of communication, and so a separate virtual connection between the broadcaster and each receiver need not be created.

This paper examines techniques for performing broadcast in PSNs and analyzes a particular one in detail. The ARPANET [Roberts72, McQuillan72] will be used as the model for PSNs.

There appear to be two ways of performing broadcast in PSNs so as to minimize the total amount of communication needed, thereby performing the broadcast quickly and cheaply, as well as lowering the possibility of subnet congestion. These techniques are

1. If a spanning tree with the smallest radius (cf section 2.) is embedded on the existing subnet with the initiator of the broadcast being the root (cf section 2.), then messages can be forwarded along the branches of this tree. This is the fastest way of performing broadcast initiated by a host connected to the root. The number of transmissions in a subnet having N nodes is N - 1. Figure 1 shows two such spanning trees for a subnet in which the cost of every edge is the same.

   Such a broadcast scheme can be implemented by laying N such spanning trees on the subnet; one for each initiator. Minimum delay routing algorithms, however, attempt to do precisely the same thing. Hence, if the subnet has a multi-destination routing scheme, then broadcast is just an extension of the inherent routing mechanism.

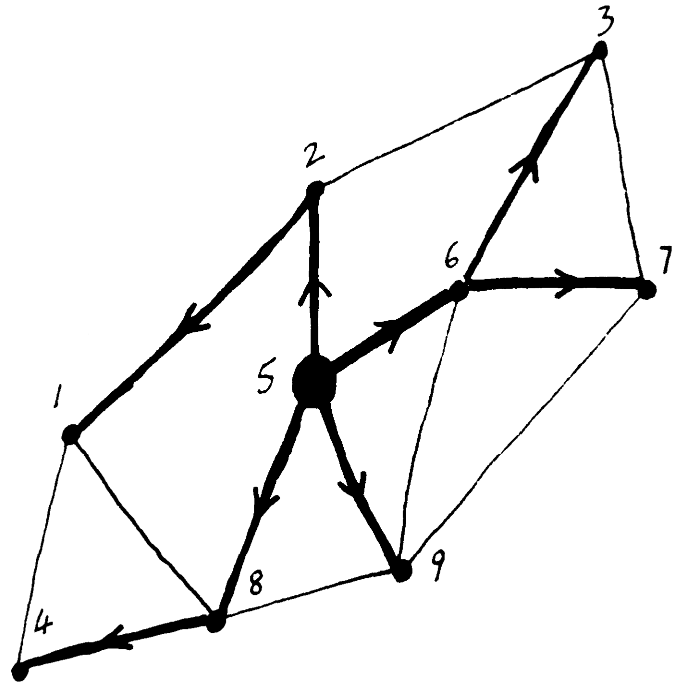


FIGURE 1a MINIMUM RADIUS SPANNING TREE WITH NODE 5 AS THE ROOT

BRANCH

EDGE

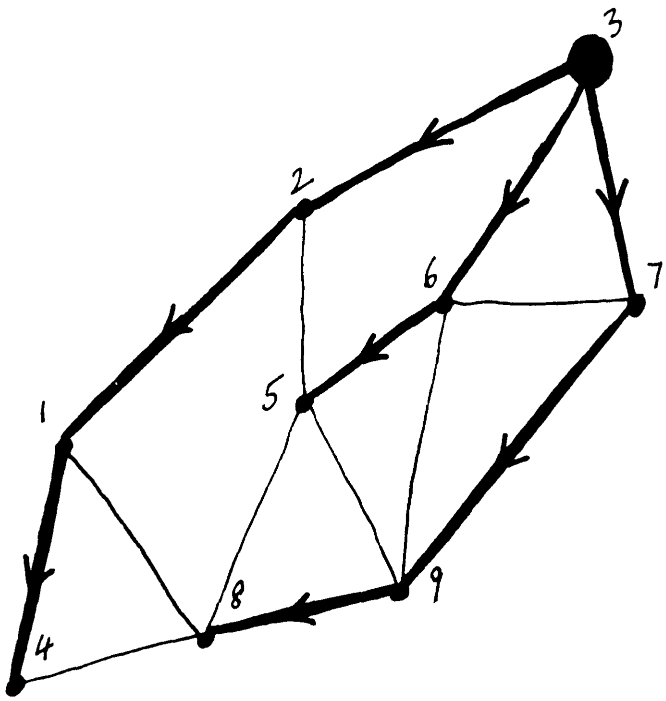


FIGURE 1b MINIMUM RADIUS SPANNING TREE WITH NODE 3 AS THE ROOT

In a subnet that has a multi-destination routing scheme, if the optimal route from one node to (say) two others included common links, then only one packet is transmitted over the common links. Broadcast is a special case of multi-destination routing, in which the destination includes all possible recipients. Routing algorithms that may be used in PSNs [McQuillan74] would then remain unchanged, but the headers of packets exchanged between switching nodes would have to be designed to carry multiple destination information, and the forwarding function of the switching node would have to be sensitive to this.

2. If a minimum spanning tree was embedded on the existing subnet topology, then any node on this minimal spanning tree could initiate a broadcast and the packets would be forwarded along this tree to all destinations. Such a technique results in the minimum transmission of packets (N - 1, in a subnet with N nodes). The time for completing the broadcast is a function of where it was initiated, as in some cases, some of the transmissions could take place concurrently. The worst case time for completing the broadcast is a function of the diameter (cf section 2.) of the minimal spanning tree. This technique assumes, of course, that the cost of communication on a branch of the minimal spanning tree is same in both directions. This is not true, in general, for PSNs, but is not a bad approximation as it could be defined as the average of the two costs. Figure 2 shows the communication subnet of a PSN with the embedded minimal spanning tree. If broadcast was initiated from a host connected to node 1, then a packet would be transmitted along each of the minimal spanning tree branches in the

directions shown in the figure. Note that all the edges in the subnet do not have the same cost.

It might be argued that if all hosts broadcast very often, then the edges comprising the minimal spanning tree would become very congested. We know that for a small number of broadcasts such a technique is preferable, and feel that even for a large number of broadcasts it will still be suitable. This feeling is based on the fact that if there were no special broadcast routing scheme, then by having a separate transmission to each destination, far more congestion would be introduced. Of course, if the minimal spanning tree were able to reconfigure itself dynamically to changing load conditions then such a technique is far more suitable. We are currently formulating an algorithm to do this. The minimal spanning tree routing scheme is very simple and may be slower (for some broadcasts) than the one in which messages are propagated along the branches of the smallest radius spanning tree. The amount by which it is slower depends on the diameter of the minimal spanning tree, the largest radius of the multi-destination spanning trees, and the pattern of broadcasts. We are also modelling this dependency more precisely.

The rest of this paper describes a distributed algorithm for constructing minimal spanning trees in computer-communication networks, in which there is no one source of control. This algorithm is both asynchronous and concurrent in its operation. Conditions under which this algorithm functions correctly will be derived, and alternatives proposed where it does not. Such an algorithm has applications in distributed operating systems as described earlier, and in communication networks like the Packet Radio Network (PRNET) [Kahn75, Frank75] in

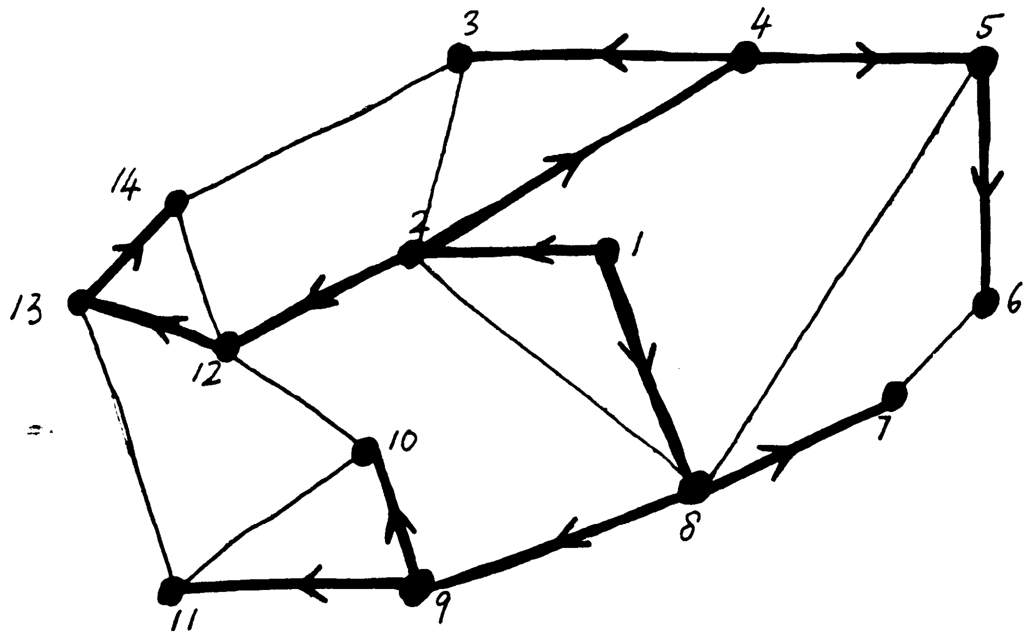


FIGURE 2 BROADCAST ALONG THE MST* INITIATED AT NODE 1

* ASSUME EDGE COSTS ARE SUCH THAT THIS IS THE MST

which the packet radio repeaters must configure themselves into a minimal spanning tree when randomly placed in an operating environment. Minimal spanning tree routing also appears to have application in the design of adaptive routing algorithms, since the branches of the minimal spanning tree could be used to transmit delay estimates to all nodes, rather than using the hop by hop refinement technique [McQuillan74].

Section 2 reviews construction principles for minimal spanning trees, and section 3 proposes a model by which these trees can be constructed in a distributed environment. Sections 4 and 5 discuss the distributed algorithm in detail.

2. Construction Principles for Minimal Spanning Trees

In this section we review definitions and construction principles of minimal spanning trees. A network is composed of a set of nodes and a set of edges that connect pairs of nodes and have a cost associated with them. The Minimal Spanning Tree (MST) of such a network is a subset of the edges such that there exists a route between every pair of nodes, and the sum of the costs is a minimum. The edges in this MST will be called the branches of the MST.

In graph theoretical terms the MST problem can be stated as follows. Consider a connected, undirected graph, G, with vertex set V, and edge set, E (E is a subset of VxV); a spanning tree is a subset of E, such that there is a unique path between any two vertices in V. Suppose there is a cost associated with every edge in E; a minimal spanning tree of G is a spanning tree of G that minimizes the sum of the cost of the edges. [Bentley75].

The path between any two nodes in a spanning tree is the sequence of edges of the spanning tree that must be traversed to get from one node to the other. The cost of a path is the sum of the edges comprising the path. If the cost of a path is larger than that of another, then that path is said to be longer than the other. The diameter of a spanning tree is the cost of the longest path in the spanning tree. The radius of a spanning tree, relative to a node called the root is the cost of the longest path from the root.

Bentley and Friedman [Bentley751 briefly review existing techniques for the construction of MSTs and propose fast algorithms for the construction of MSTs in multidimensional coordinate spaces. These algorithms are of the order of NlogN, where N is the number of nodes. A complete bibliography on the subject (upto 1974) can be found in [Pierce75]. The construction principles for MSTs were first formalized by Prim [Prim57] and are applicable to networks for which the edge costs (length, distance, delay) need not be distinct and could be anything, and thus need not be consistent with Euclidean geometry.

The networks of interest to us will be the general class of networks studied by Prim.

2.1 Prim's Principles

Prim (1957) suggested two principles for constructing MSTs. We paraphrase some of his definitions, construction rules, and conditions. The principles assume that the construction process is sequential. An isolated node is a node to which, at a given stage of the construction, no connections have yet been made. A fragment is a subset of nodes connected by edges (which will become branches) between members of the subset. An isolated fragment is a fragment to which, at a given stage of construction, no external connections have been made. The distance (cost) of a node from a fragment of which it is not an element is the minimum of its distances (costs) from each of the individual nodes comprising the fragment. A nearest neighbor of a node is a node whose distance from the specified node is at least as small as that of any other. A nearest neighbor of a fragment, analogously, is a node whose distance from the specified fragment is at least as small as that of any other.

Prim proved that a MST could always be constructed by following the following two principles.

Principle 1 (P1): Any isolated node can be connected to a nearest neighbor.

Principle 2 (P2): Any isolated fragment can be connected to a nearest neighbor by a shortest available edge*.

These principles were based on two necessary conditions.

Necessary Condition 1 (NC1): Every node in a MST must be connected to at least one nearest neighbor.

Necessary Condition 2 (NC2): Every fragment in a MST must be connected to at least one nearest neighbor by a shortest available edge.

*The nearest neighbor of a fragment may be connected to the nodes of the fragment by more than one edge. Usually the process of determining the nearest neighbor of a fragment will involve examining the edge costs connecting nodes within the fragment to nodes outside it, and so the shortest available edge will easily be determined.

2.2 Existing Algorithms

Most existing algorithms use P1 and P2 to create an isolated fragment and then increase the number of nodes in the fragment until it becomes a MST. The primary concern has been how to structure the data so that it is possible to quickly determine the shortest edge by which an isolated fragment can be connected to a node outside it. This is of great importance if a fully connected network having a large number of nodes is under study. All these algorithms are sequential; there is no concurrency in growing many isolated fragments. The multi-fragment algorithm [Bentley75] is sequential in its operation.

The goal of this paper is to describe a concurrent, asynchronous algorithm to create an MST. Such an algorithm is desirable not necessarily for the increased speed of execution (which we expect), but also because it ensures that there is no one source of control. Such algorithms are ideally suited to computer-communication networks. The algorithm may also be used for constructing MSTs for other applications using a multiprocessor computer, such as the Pluribus [Ornstein75].

3. Distributed MST algorithms

A distributed algorithm consists of a program executing in each of the nodes such that, when all the programs terminate, the result would be a MST connecting the nodes. Every node will know which of the edges connected to it are branches of the MST. It will be necessary for the nodes to communicate with their neighbors or some other node by means of messages. Properties of such algorithms that are of interest include:

(i) Does shared information between nodes have to be locked when modified?

(ii) What form of synchronization is required between the nodes?

(iii) Are there any special initial conditions?

(iv) Does the algorithm work only for certain combination edge costs?

(v) In a network environment can the algorithm account for some of the nodes going down, edges breaking, or new nodes coming up?

In the discussion of the algorithm, and in proving its correctness, we will constantly map the state of the evolving MST to that of a MST being constructed by conventional sequential methods.

3.1 The Basic Model

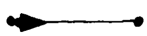
In order to prove that any algorithm for constructing MSTs works, it is sufficient to show that every operation performed is identical to P1 or P2, and that the algorithm terminates.

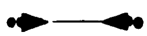
The underlying philosophy of the distributed algorithm for constructing a MST is based on NC1, which states that every node must be connected to at least one of its nearest neighbors. Hence every node knows which neighbor to form a branch with. However, the result of such an action by every node will create a MST only in some cases. In general, such an action will produce a number of fragments that must be connected together appropriately. The distributed algorithm must discover that such a fragment has been created and then choose an appropriate edge to connect the fragment to other such fragments without introducing any cycles in the graph.

We now introduce some definitions, and prove some simple properties of the model. This is mainly to provide a framework and vocabulary for the treatment of the algorithm which will follow.

3.2 Definitions

Figure 3a shows a network, in which every edge has a cost associated with it. The MST for this network is shown in figure 3b. Notice that some of the branches of this MST have been marked. The markings have the following interpretation:

 Such a branch is called a singly marked branch. This branch is part of the MST since it connects the node from which the arrow emanates to its nearest neighbor (by virtue of NC1).

 Such a branch is called a doubly marked branch. It connects both nodes to their nearest neighbors.

 This branch is unmarked.

In figure 3b, edges BD, CF, GF, FJ, NM, EI, HK and PO are singly marked. Edges AD, IK, LO and JM are doubly marked while DE, EF and IL are unmarked.

The largest fragment composed only of marked branches (singly or doubly) will be called a Marked Fragment (MF). Notice that MFs are connected by unmarked branches to form larger fragments until the MST is formed. In figure 3b, unmarked branch DE connects marked fragments {A, B, D} and{ H, K, I, E}.

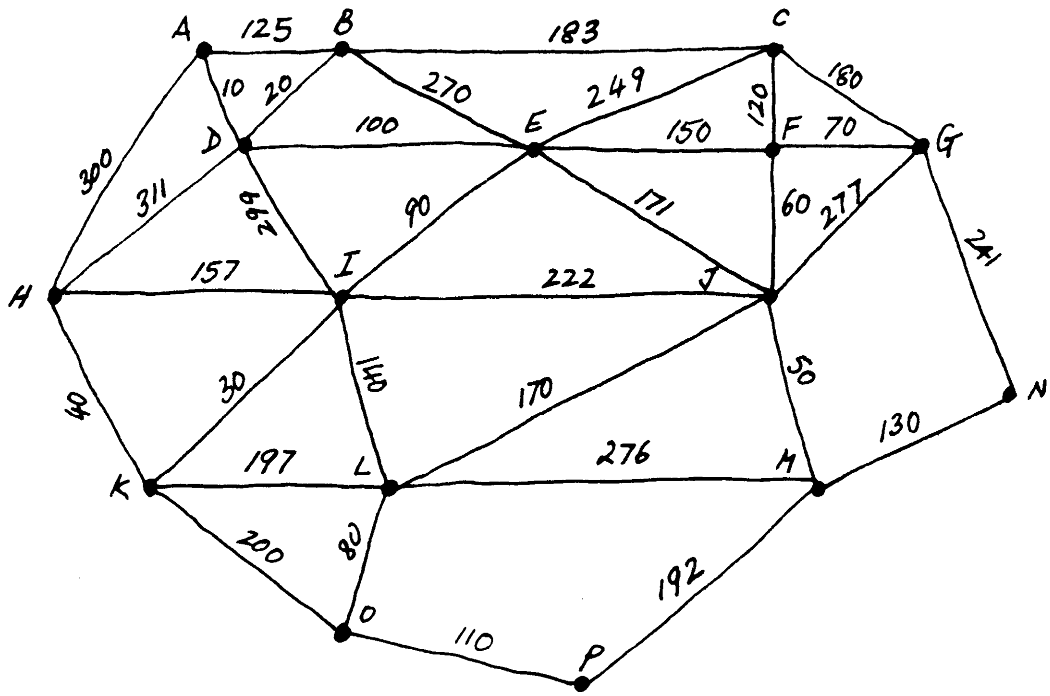


FIGURE 3a A NETWORK

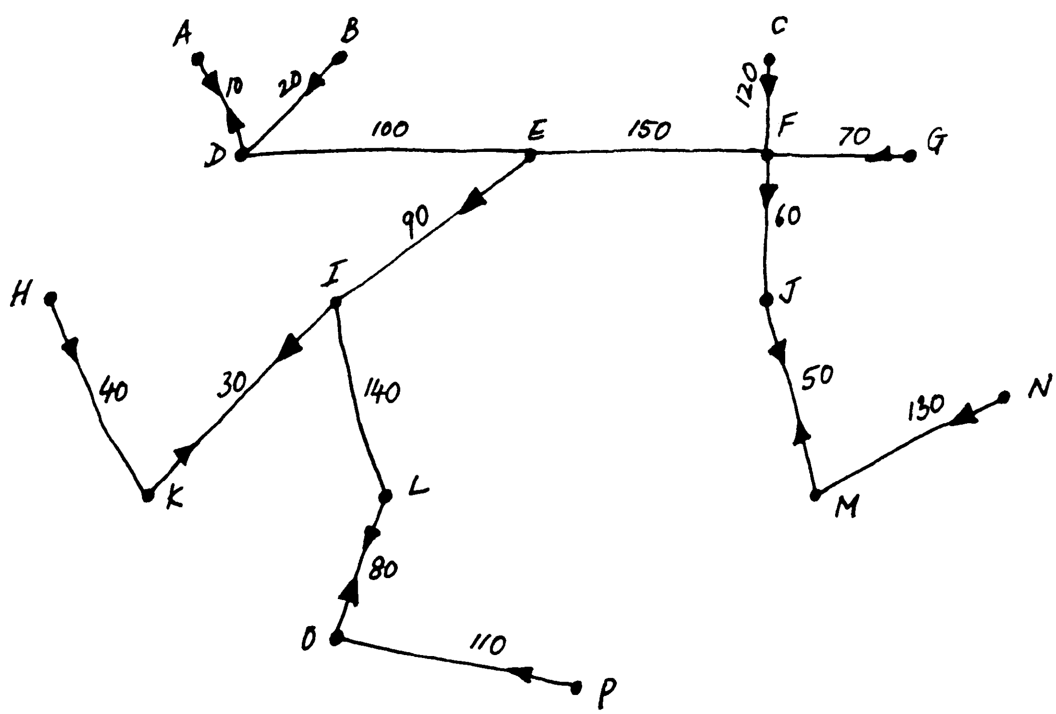


FIGURE 3b THE NETWORK'S MST

We now state and prove some simple properties of these MSTs.

We know that a network with N nodes has a MST with N-l branches.

Theorem 1 : In a MST, the number of MFs equals the number of doubly marked branches, and each such fragment contains exactly one doubly marked branch.

Proof: By definition every MF is a MST for that node subset. Since the number of marked branches is equal to the number of node in that set minus one, it follows that one branch must be doubly marked. Hence, each MF has one and only one doubly marked branch. Therefore the number of MFs is equal to the number of doubly marked branches of the complete MST. ■

Corollary 1.1: Every MST has at least one doubly marked branch

Proof: The smallest number of MFs in a MST is one, and therefore the proof follows from Theorem 1. ■

In a network that does not have distinct edge costs a number of MSTs are possible. There may be different ways of creating MFs in each case and so the number and identity of the doubly marked branch will vary.

Theorem 2: In a MST, the number of unmarked branches is equal to one less than the number of MFs.

Proof: For a given MST let NMF be the Number of Marked Fragments. Let n(i) be the number of nodes in the ith MF.

Since each MF is a MST for its node subset, the number of branches in the ith MF is $n(i) - 1$. Therefore, the number of marked branches in the complete MST is equal to

$$\sum_{i=1}^{NMF} [n(i) - 1]$$

$$= N - NMF$$

The total number of branches in the complete MST is equal to $N - 1$, and therefore the number of unmarked branches in the MST is equal to

$$N - 1 - (N - NMF)$$

$$= NMF - 1$$

and hence the theorem is proved. ■

A chain is a node subset (containing one or more nodes) connected by edges between members of the subset, such that each edge connects a node to its nearest neighbor (and hence is also a branch). Edges are unique to a node, i.e. an edge can not connect two nodes to each others nearest neighbors. Such a chain is a fragment, and a MST for the node subset. The chain will be said to have one active node - the node that will connect itself to its nearest neighbor and still keep the fragment a chain. Chains only have branches which are singly marked. In figure 3b some of the chains and their active nodes are {GF; F active}, {G; G active), {GF, FJ; J active), {CF, GF, FJ; J active} and {CF, GF; F active).

Notice that there is a certain monotinicity among the costs of branches in a chain. For every node in the chain, the cost of the branches incident at the node (as determined by the markings on the branch) is larger or equal to the cost of the branch leaving the node (there is only one). This fact will be proved in the following theorem.

Theorem 3: The cost of the potential branch from the active node of the chain must be less than or equal to the cost of the branches in the starting chain.

Proof: This theorem is proved by induction.

If the chain consists of only one node (which is also active) then the cost of the potential branch must be less than those already in the chain (the null set).

Now assume that the chain has n branches satisfying this property. The active node has at least one branch incident at it. The active node has an edge to its nearest neighbor. The cost of this edge can be less than or equal to that of the lowest cost incident branch, but not more otherwise the node at the other end of the potential branch would not be the active node's nearest neighbor. ●

Corollary 3.1: If the edge costs are distinct, then the branch out of the active node of a chain has a cost less than that of any branch in the starting chain.

Proof: The proof is identical to that for Theorem 3, except that since edges have distinct costs, it can never be that the cost of the potential branch out of an active node is equal to that of a branch incident to the active node in the starting chain. ■

If the active nodes of two chains decide they are each others neighbors, then the two chains merge, and this branch becomes a doubly marked branch of the resulting MF that these two chains are part of. The resulting fragment is no longer a chain.

When MFs connect to each other by an unmarked branch, the resulting fragment will be called a Minimal Spanning Subtree (MSS). A MSS becomes a MST when it contains all the MFs. The active node of a MF or a MSS is the node from which the unmarked branch to another MF or MSS will emerge. A MST has no active node because there are no more branches to create.

We now prove some simple properties for unmarked and doubly marked branches. The proofs will be made for networks with distinct edge costs. The theorems will be valid for networks with this restriction removed except that the strict inequality will be replaced by a weaker inequality.

Theorem 4: For a network with distinct edge costs, the doubly marked branch of a MF has the lowest cost among the branches of that fragment.

Proof: The two nodes on either end of the doubly marked branch are active nodes of two chains which have all the nodes of the MF contained within them. The potential branch from these active nodes have a cost less than that for branches in their respective chains (from Corollary **3.1)**. The cost of potential branches is the same since they are the same branch - a doubly marked branch. Hence the cost of a doubly marked branch is less than that of any other branch in the MF. ■

Theorem 5: For a network with distinct edge costs, the cost of an unmarked branch connecting two MFs is larger than that for the doubly marked branch of either MF.

Proof: The unmarked branch is connected to a node in the MF. This node is also connected to a marked branch in the MF and so the cost of the unmarked branch is greater (since edge costs are distinct) than that of the marked branch. From Theorem 4 it follows that the cost of this marked branch is greater than or equal to that of the doubly marked branch of the MF. Hence the cost of the unmarked branch is larger than that of the doubly marked branch. Since this applies to both MFs connected by the unmarked branch, the theorem is proved. ■

These definitions and proofs are useful in understanding how the distributed algorithm for constructing a MST works, since the algorithm revolves around the ideas of concurrently creating MFs and having them grow into MSSs until the MST results.

4. Statement of the Algorithm

We now describe a distributed algorithm for constructing a MST in a network with distinct edge costs. In the next section we show how this algorithm can be extended to construct a MST in a network where the edge costs are not distinct.

Since the edge costs are distinct, the MST is unique [Kruskal56].

The basic philosophy of the algorithm is that each node must independently find its nearest neighbor and make the edge connecting it to that neighbor into a branch of the MST. The node then sends off a message to the neighbor informing it of this construction. Two nodes may realize that they are connected by a doubly marked branch. This is when the core of a MF is formed. This must grow into the MF. Such MFs will connect to other MFs or MSSs until a MST is created. Since the edge costs are distinct, the MST is unique. We will show in detail how MFs connect to other MFs or MSSs and also that no cycles are introduced by the asynchrony and concurrency of the computation.

We now introduce some more terminology. A node is said to be the master if it decides from which node of the fragment a branch should be created to a node lying outside the fragment. The node that actually makes the construction will become active. In a MF there is only one node that can be master. Initially there are no masters. When a doubly marked branch gets created, one of the two nodes at either end unambiguously becomes master. We show later how this decision can be made. When two MFs get connected by an unmarked branch, there may be two potential masters (one in each MF). One of the masters unambiguously relinquishes control to the other, who then determines which node (of the fragment it has knowledge about) becomes active. The result of all this is a MST!

Since there is one unique unmarked branch connecting two MFs, there is never any ambiguity in choosing it. Hence a race condition can not arise, where the two MFs choose different edges as branches connecting each other, thereby creating a cycle.

Every operation described so far has been consistent with Prim's Principles. We will show this more precisely a little later, and will now proceed to describe the algorithm formally.

4.1 State Information at Each Node

The statement of the algorithm will assume that each node has a set of state variables. These consist of the node state, information about each of the edges this node is part of, and a list of all nodes that are part of the fragment as seen by this node. This list contains for each node in the fragment*, edges that connect them to nodes outside the fragment and their costs.

4.1.1 The Node State

The variable NODESTATE is equal to inactive, active or master. This variable determines what the algorithm should do when it gets messages from other nodes.

4.1.2 Edge Information

The node has a descriptor for each edge from that node. This information is called EDGEINFO and consists of the following entries:

SOURCE - The source node of this edge. It is the identity of this node.

DEST - The destination node of this edge. It is equal to the identity of the node at the other end of this edge.

COST - The cost associated with this edge.

BRANCH - A boolean, which if true indicates that this edge is a branch of the MST.

MYMIN - A boolean, which if true indicates that this edge is a branch and was marked by this node, since it is the MINimum cost edge at this node.

HISMIN - A boolean, which if true indicates that the this edge is a branch and was marked by DEST since it was the MINimum cost edge at that node.

ICON - A boolean, which if true indicates that this node made this edge into a branch but did not mark it. This is because the node CONnected the fragment, it is part of and has know1 edge of, to the fragment's nearest neighbor.

HECON - A boolean, which if true indicates that DEST made this edge into a branch but did not mark it. This is because DEST was CONnecting the fragment, it has knowledge of, to the fragment's nearest neighbor.

4.1.3 The Fragment State

A data structure called the FRAGSTATE represents the state of the fragment as seen by this node. Conceptually, it could be viewed as a table indexed by nodes which lie in the fragment. For each such entry, there is a chain of entries identifying edges which connect this node to nodes outside the fragment, and their cost. Note that some of these edges could be branches, since the node at which this data structure resides may not know to what other nodes the node at the other end of the branch is connected to, and so can not include the node in the fragment state. Any suitable data structure which permits a fast search will do. Note that any node already in the fragment can not be part of an edge for another node in the fragment.

4.2 Internode Communication

Internode communication is achieved by sending messages called SIGNALS. Signals have a number of parameters. A signal can be sent to a node that is a neighbor, or to a node that is part of the same fragment.

FROM - The node from which the signal originated.

TO - The destination of the signal.

FRAGSTATE - The fragment state at the node at the time the signal was created.

EDGEINFO - The descriptor for the edge that is being made into a branch.

COMMAND - This causes a particular action at the destination of the signal. If the command is "connect", it implies that a marked branch is being created. EDGEINFO must be present. If the command is "master", it implies that the destination node is to become master. If EDGEINFO is also present then a branch (potentially unmarked) is also being created. If it is not then the command acts as a transfer of master control.

4.3 Associated Routines

There are some special routines at each node. MERGEFRAGSTATE merges the fragment state received in a signal with the fragment state already present at the node. Merging consists in adding nodes not already part of the fragment and deleting edges whose nodes now lie within the fragment.

A routine called MERGEDGEINFO merges the edge information received in the signal with that contained for this edge at the node.

DECIDE is a routine that determines which of two nodes should become master. Relative node numbering could be used as an unambiguous decision. More esoteric techniques could be used which may help the algorithm execute faster. For example, both nodes know which edges the other is part of (since both nodes just exchanged FRAGSTATEs). The node that becomes master is the one that has a lower cost edge excluding the one that connects both together.

ANYNEIGHBOR is a routine which examines FRAGSTATE and determines which node (if any) should become master. If ANYNEIGHBOR returns true, then the identity of this node is returned in MASTERNODE, and the identity of the node at the other end of the edge from MASTERNODE in DESTNODE. The edge determined by (MASTERNODE, DESTNODE) connects this fragment to its nearest neighbor.

4.4 The Main Program

This is the main program. It consists of a main loop and a procedure call. Both use the data structures and routines defined in the previous sub-section. The program will be written in an ALGOL-like language.

```
procedure TRANSFERMASTERCONTROL;
begin
comment - This procedure examines FRAGSTATE to find the MASTERNODE
and the DESTNODE. If the MASTERNODE is itself, then the node converts
the edge determined by (MASTERNODE, DESTNODE) into a branch if it is not
already one; if it is a branch then DESTNODE is signalled to become master.
If a branch was being created it does not have MYMIN set
since it is not the node's nearest neighbor.
If the MASTERNODE is not itself, then that node is told to become master;

if ANYNEIGHBOR
then begin
    if [MASTERNODE = this node]
    then begin
        if [For this edge, BRANCH = true]
        then SIGNAL(ME, DESTNODE, FRAGSTATE, null, MASTER)
        else begin
            comment - Convert this edge into a branch;
            NODESTATE := ACTIVE;
            [For this edge, BRANCH := ICON := true];
            SIGNAL(ME, DESTNODE, FEUGSTATE, EDGEINFO, MASTER);
            end
        end
    else SIGNAL(ME, MASTERNODE, FRAGSTATE, null, MASTER);
    end;
end TRANSFERMASTERCONTROL;
```

```
procedure MAINLOOP;
begin
comment - This is the main loop of the program;

comment - Local initializations;
[Determine the cost of all possible edges at this node, and for
each create a descriptor EDGEINFO. Set up the parameters of EDGEINFO
appropriately, with BRANCH := MYMIN := HISMIN := ICON := HECON := false];

[Build FRAGSTATE];

comment - Convert an edge into a branch using NC1;
NODESTATE := ACTIVE;
if ANYNEIGHBOR then
    begin
    [For this edge, BRANCH := MYMIN := true];
    comment - Signal the node at the other end of the branch;
    SIGNAL(ME, DESTNODE, FRAGSTATE, EDGEINFO, CONNECT);
    end;

NODESTATE := INACTIVE;

comment - Now wait for for a signal from other nodes;
LOOP:begin
    [Wait for a signal];
    comment - A signal just arrived, so continue;

    MERGEFRAGSTATE;
    case [The COMMAND field of this signal] of
        begin
            begin
```

```
comment - COMMAND = CONNECT;
MERGEDGEINFO;
if [For this branch, MYMIN = true] then
     begin
     comment - This is a doubly marked branch;
     NODESTATE := MASTER;
     if DECIDE then TRANSFERMASTERCONTROL;
     NODESTATE := INACTIVE;
     end;
end;

begin
comment - COMMAND = MASTER;
NODESTATE := MASTER;
if [For this signal, EDGEINFO := null]
then begin
     comment - The node is not required to change an
     edge into a branch, but just to find the right master;
     TRANSFERMASTERCONTROL;
     end
else begin
     comment - Not only does this node have to
     find the right master, but it has also been
     told about a new branch. This may be an
     unmarked branch;
     MERGEDGEINFO;
     if [For this branch, ICON := true]
     then begin
```

```
                      comment - Both nodes of this branch
                      converted the edge into an unmarked
                      branch. Resolve who is master;
                      if DECIDE then TRANSFERMASTERCONTROL;
                 else TRANSFERMASTERCONTROL;
                 comment - Just find the right master;
                 end;
            NODESTATE := INACTIVE;
            end;
        end;
    end;
repeat LOOP;
end MAINLOOP.
```

4.5 Analysis of the Algorithm

The analysis of the algorithm is probably the most difficult part. We put off determining its complexity for the present, and just prove that it does in fact construct the MST. The underlying basis for its correct functioning is that the resulting MST is unique, and the premise that every edge made into a branch by a fragment is consistent with Prim's Principles. We will now justify the premise.

Recall the following properties of MSTs and the algorithm:

(i) Every node creates a branch out of the edge that is of minimum cost incident to itself. The message indicating this may take a while getting to the node at the other end of the branch.

(ii) Marked Fragments are connected together by unmarked branches to create the MST.

Every node starts off by creating a branch out of edges incident to it using P1. The node informs its neighbor at the other end of this branch. This message may incur a delay before arriving at its destination, and in the meantime the generator of the message is free to continue processing.

Every node now waits for messages.

If a message arrives announcing the establishment of a singly marked branch, then the node checks to see if it too had marked this branch. If not, then the node updates its data structures and continues to wait for other messages (should there be any).

If this branch turns out to be a doubly marked branch, then the core of a MF has been created, and one of the two nodes unambiguously becomes master. There may be many such cores in creation in the network. This event is of great importance in the algorithm. The node that is master of this MF must now grow this MF into a MST using P2. In other words, the master node is in search of an "unmarked branch" that will connect this MF to another MF or MSS. The decision on which edge to convert to a branch is based on the node's current information of the fragment. We know that the resulting MST is unique since the edge costs are distinct, and that a fragment must be connected to its nearest neighbor (P2). Hence this branch is unique, and so even with the asynchrony in the operation of the algorithm, the decision of the active node is always correct. Note that this is true even when the signals take different amounts of time to be successfully transmitted. This is elaborated below.

In quest of this "unmarked branch" the node may pick an edge such that the node at the other end is part of the same MF. This is possible since the message from that node announcing the creation of the singly marked branch may not have yet arrived. Such an action is not harmful and is in fact important. Master control will be transferred to the new node, which will now grow the MF with the help of more complete fragment information, and master control will propagate until the "unmarked branch" to another MF or MSS is found.

A node that is master may even decide that an edge that has already been made into a branch (but still exists in the fragment state) connects the fragment to its nearest neighbor outside the fragment. The node just transfers master control to that node since it may have a more accurate view of the fragment and can make a better decision. The node which transfers master control can not pick another edge to convert into a branch since it is not the lowest cost edge and can easily create a cycle.

Note that a node that is active may convert an edge into the "unmarked branch" without knowing what its complete MF looks like. This is not harmful since MF branches always consist of constructions based on P1, and messages notifying neighbor nodes of this construction will eventually arrive.

Two active nodes may decide to make an edge into an unmarked branch simultaneously, in which case one of them unambiguously relinquishes control to the other, and the master grows this MSS.

Note that when the algorithm starts, there may be many nodes that are master nodes, but eventually this number will decrease until there is only one. This one will eventually determine that there are no more nodes lying outside this fragment, and will thus conclude that the MST has been created. The program at each node is said to terminate when it receives no more messages. Of course, the node does not know if it is going to receive any more signals or not, and so if it transmits messages along the branches of the minimal spanning tree before it has been completely constructed, the messages may not get to all destinations. The proof of the fact that the algorithm terminates is based on the observation that a new signal only gets sent, (in response to one received, that has a command indicating that the node should become master) if and only if the fragment state at the node indicates that there is a possibility of still growing the fragment. Nodes which are told to become master will eventually refine their fragment states such that there will be no nodes lying outside the fragment and so no more signals will be generated.

The algorithm is thus very similar to the large class of sequential MST algorithms that use P1 once and then use P2 continuously. However, this algorithm is self synchronizing, and thus suitable in an asynchronous, concurrent operating environment.

5. Networks in which the edge costs are not distinct

The algorithm presented in the previous section constructed a MST since the edge costs were distinct, and so the decision made by an active node to convert an edge into a branch was always correct and unaffected by the asynchrony of the computation.

When the edge costs are not distinct, the asynchrony of the operation may introduce cycles, and thus will not construct a MST. To see why this is possible, consider the example shown in figure 4. Nodes 1, 2, and 3 are part of a larger network. Edges (1, 2), (2, 3) and (1, 3) are all of the same cost. It may so happen that when each node is converting an edge into a branch using P1 that node 1 chooses 2, node 2 chooses 3, and node 3 chooses 1. A cycle has resulted.

Similarly, if there are two MFs that have more than one possible unmarked branch connecting them together, then the master node in each MF may choose a different edge to convert into a branch, thus creating a cycle. Generalizing, we can say that if there is more than one edge that can be converted into a branch so as to connect two fragments together, then there is the possibility of a cycle.

Prim (1957) showed that if there are many edges of the same cost connecting a fragment to its nearest neighbor, then it did not matter which was chosen, and a MST would still be constructed.

Therefore if the network is converted into one with distinct edge costs, either implicitly or explicitly, then the algorithm presented in section 4, would be suitable since it would construct a MST. The next subsection indicates how a network can be converted into one with distinct edge costs very easily. Although we would like to create a MST with the minimum diameter, since that would reduce the maximum time for broadcast, any MST will do. We feel that by using a concurrent, asynchronous algorithm based on Prim's "greedy" algorithm, it is not possible to guarantee that the MST constructed will be the one with the minimum diameter.

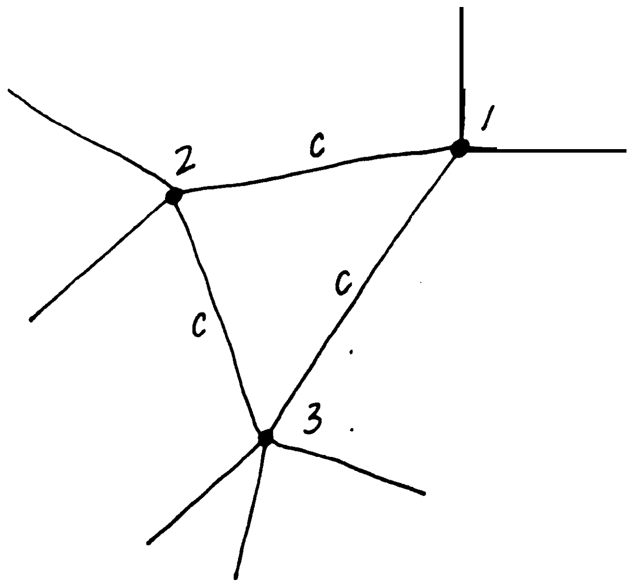


FIGURE 4. A PART OF A NETWORK WITH NON-DISTINCT EDGES. THE POSSIBILITY OF A CYCLE EXISTS.

5.1 Transforming a network into one with distinct edge costs

In this section a technique is described for converting the network into one with distinct edge costs so that the algorithm presented earlier is still usable. This technique is again distributed and is an extension of the previous algorithm, as we shall see.

Since there is only one edge connecting any two nodes in the network, and nodes have distinct identities (numbers) each edge has a unique pair of node identities associated with it. This makes it very easy to dynamically order edges with the same cost, thus transforming the network into one with distinct edge costs.

Let us assume that the edge costs are accurate to the Mth decimal place. When deciding which edge to convert into a branch, the node could modify (temporarily) the edge cost using the following algorithm.

Let C(e) be the cost associated with edge e, where e is a tuple (N1, N2), where N1 and N2 are the identities of the two nodes. Let NN be the total number of nodes in the network. Then the new value of C(e) is given by:

```
C(e)new = C(e)old + min[N1, N2]*(10↑-ceiling(logNN))*(10↑-M)
                  + max[N1, N2]*(10↑-(2*ceiling(logNN)))*(10↑-M)
```

This complicated looking formula is just adding the number got by correctly concatenating the minimum of N1 and N2, and the maximum of N1 and N2 to C(e)old, beyond the Mth decimal place.

Note that this is a distributed computation, and so the master nodes in two fragments unambiguously decide which of two edges with equal costs is the "lower" cost one. Since the computation that decides this only affects the edge cost beyond the Mth place of accuracy, the relative ordering between the edge costs has not changed, and a MST can be constructed.

This computation can be performed by ANYNEIGHBOR every time it decides to find the minimum cost edge, or only when it realizes that there are two potential edges that could become branches, thus breaking the tie. The computation need not performed explicitly by modifying the edge costs as specified by the formula, and then testing if one edge cost is less than the other. It can also be performed by examining the magnitudes of the node identities (as specified by the formula), and thereby order the edge costs without having to worry about loss of accuracy in performing the arithmetic.

6. Conclusions

An algorithm has been described that is useful for constructing a MST in a computer-communication network, or a multiprocessor. This algorithm is asynchronous and concurrent, and so can be thought of as a parallel algorithm for constructing a MST. It is believed that this is the first algorithm of its kind to construct MSTs. Networks which do not have distinct edge costs can very easily be converted into ones that do, thus making them suitable for the algorithm.

The algorithm has the following properties:

(i) EDGEINFO is a data structure that is duplicated at both nodes of the edge. This data structure reflects the state of the edge, and need not be locked when each node decides to modify it.

(ii) Synchronization between the nodes for the purpose of creating branches, and for refining the state of the fragment at each node is achieved by sending message from a node to either its neighbor or to another node in the same fragment. Transmission of messages to a node that is not a neighbor but in the same fragment, can be done by broadcasting (or relaying) it along the branches of the ?lST of this fragment. Hence there is no need for another routing scheme.

(iii) The only special initial condition is that all nodes know the cost of the edges connecting them to other nodes, and the identities of those nodes. Each node must also know the maximum number of nodes in the network, and must maintain the edge costs with the same degree of precision.

(iv) The algorithm is able to construct a MST in a network that has no constraint on the combination of edge costs.

(v) The algorithm can not incrementally account for nodes going down, edges breaking or nodes coming up. The MST has to be recomputed.

We are in the process of determining the complexity of this algorithm, and formalizing an adaptive algorithm that dynamically reconfigures a MST when edge costs change.

7. Acknowledgements

I would like to thank B. Ramakrishna Rau for the many discussions and exchange of ideas that helped crystallize this algorithm, and Phil

Spira for suggesting an idea that led to the technique for handling networks with non-distinct edge costs.

8. References

[Abramson70] N. Abramson, "THE ALOHA SYSTEM - Another alternative for Computer Communication," Fall Joint Computer Conf., 1970, AFIPS Press, pp. 281-285.

[Abramson73] N. Abramson, "Packet Switching with Satellites," National Computer Conf., 1973, AFIPS Press, pp. 695-702.

[Bentley75] J. L. Bentley and J. H. Friedman, "Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces," SLAC Technical Report SLAC PUB-1665, Stanford University, December 1975.

[Cosell75] B. P. Cosell, P. R. Johnson et. al., "An Operational System for Computer Resource Sharing," Proc. Fifth Symposium on Operating Systems Principles, November 1975, (available as ACM Operating Systems Review, Vol. 9, No. 5, pp. 75-81).

[Crocker75] S. D. Crocker, "The National Software Works: A New Method for Providing Software Development Tools using the Arpanet," Consiglio Nazionale delle Ricerche INSTITUTO DI ELABORAZIONE DELLA INFORMAZIONE Meeting on 20 years of Computer Science, Pisa, June 1975.

[Dalal76] Y. K. Dalal, "Distributed File Systems," Presented as a working paper at the Berkeley Workshop on Distributed Data Management and Computer Networks, May 25-26 1976.

[Farber72] D. J. Farber and K. C. Larson, "The Structure of a Distributed Computing System - The Communication System," Proc. of the Symposium on Computer-Communications Networks and Traffic, Polytechnic Institute of Brooklyn, April 1972, pp. 21-27.

[Farber72a] D. J. Farber and K. C. Larson, "The Structure of a Distributed Computing System - Software," Proc. of the Symposium on Computer-Communications Networks and Traffic, Polytechnic Institute of Brooklyn, April 1972, pp. 538-545.

[Frank75] H. Frank, I. Gitman and R. VanSlyke, "Packet Radio Network Design - System Considerations," National Computer Conf., 1975, AFIPS Press, pp. 217-231.

[Kahn75] R. E. Kahn, "The Organization of Computer Resources Into a Packet Radio Network," National Computer Conf., May 1975, AFTPS Press, pp. 177-186.

[Kruskal56] J. B. Kruskal Jr., "On the Shortest Spanning Subtree of a graph and the Traveling Salesman Problem," Proc. Amer. Math. Soc., 7, 1956, pp. 48-50.

[McQuillan72] J. M. McQuillan, W. R. Crowther, B. P. Cosell, D. Walden and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network," Proc. Fall Joint Computer Conf., 1972, AFIPS Press, pp. 741-754.

[McQuillan74] J. M. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks," BBN Report No. 2831, May 1974.

[Metcalfe75] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," Xerox Palo Alto Research Center Technical Report CSL 75-7, November 1975.

[Ornstein75] S. M. Ornstein, W. R. Crowther, M. F. Krayley, R. D. Bressler, A. Michel, F. E. Heart, "Pluribus - A reliable multiprocessor," National Computer Conf., May 1975, AFIPS Press, pp. 551-559.

[Pierce75] A. R. Pierce, "Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree, and Related Circuit Routing Problems (1956-1974)," Networks, Vol. 5, 1975, pp. 129-149.

[Prim57] R. C. Prim, "Shortest Connection Networks and Some Generalizations," Bell Systems Tech. J., November 1957, pp. 1389-1401.

[Roberts72] L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 543-549.

[Thomas73] R. H. Thomas, "A Resource Sharing Executive for the ARPANET," Spring Joint Computer Conf., June 1973, AFIPS Press, pp. 155-163.