2.6.1 The Basic Model

Recall the abstract parallel MST algorithm described in section 2.3. Fragments (subtrees of the MST) connect to other fragments to create larger fragments until the one remaining fragment spans the entire set of nodes. Various implementations of this algorithm impose constraints on which fragments can connect to other fragments, the node within a fragment that makes the decision to connect to another fragment, and the form of internode communication and transfer of control. The algorithm described in section 2.4 transferred master control from node to node, while the one in section 2.5 determined which node should become the root when two fragments were connected to one another by the minimum cost edge, to form a larger fragment.

The algorithm to be described in this section uses the old spanning tree to systematically pass control around so as to transform this spanning tree into the MST. A tree spanning a network with N nodes, has N-1 branches. The algorithm requires that each node (save one) change one of these N-1 branches (one that is incident to it) into a branch of the new MST, thereby completing the transformation. The terminology used to describe the algorithm is similar to that used in section 2.4. We will repeat definitions when appropriate.

A node is said to be the leaf of the old spanning tree if it is connected to the old spanning tree by only one branch. In figure 2.4b A, C, F, H, K, M, N and P are the leaf nodes of an old spanning tree for the network in figure 2.4a. A node is said to be master if it decides

from which node of the new-fragment a branch should be created to a node lying outside the fragment. The node that actually makes the construction will become active. When a node becomes a leaf, it must remove the old-branch connecting it (and the fragment of the new MST) to the old spanning tree, and replace it with a new-branch that connects the new-fragment to its nearest neighbor. In this way the old spanning tree is converted into a new MST.

A node becomes master when it becomes a leaf node. The last branch of the old spanning tree is removed and a new one found. The master node may transfer master control to some other node in the fragment in search of the edge that connects this fragment to its nearest neighbor. This may recur till a node becomes active. A new-branch is created and master control disappears. However, if two active nodes convert the same edge into a branch, then one of them must unambiguously become master again. This is because a master node and therefore an active node was the result of removing an old-branch and replacing it with a new one. Each active node must create a branch, and so if two of them create the same branch, there is one less new-branch and one of the two active nodes must becomes master again.

Initially, the only fragments that can compute are those that consist of single nodes that are leaves of the old spanning tree. These nodes, asynchronously and concurrently, remove the branch that connects them to the old spanning tree and create a new-branch that connects them to their nearest neighbors. As a consequence of such an action by leaf nodes, other nodes in the network will become leaf nodes and the process

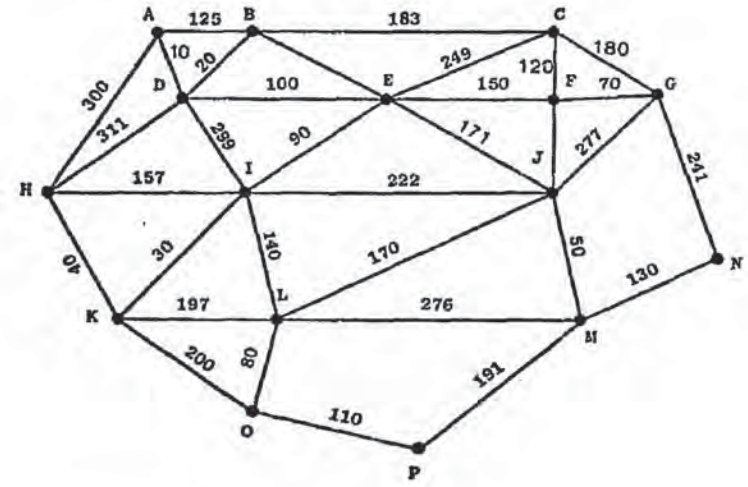


Figure 2.4a. A NETWORK.

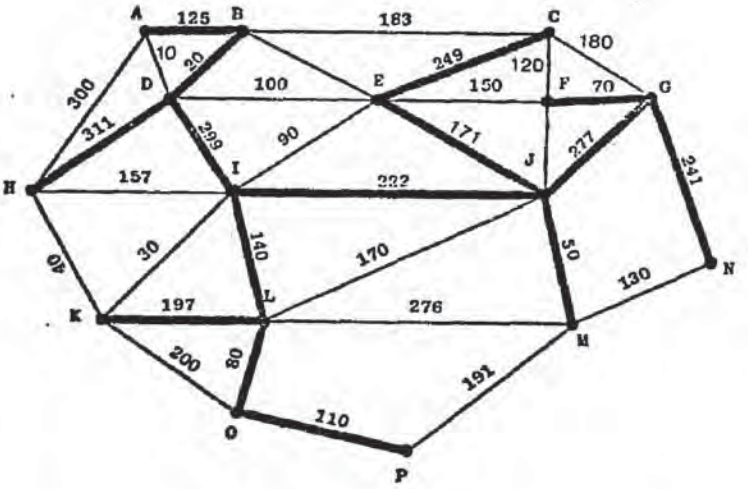
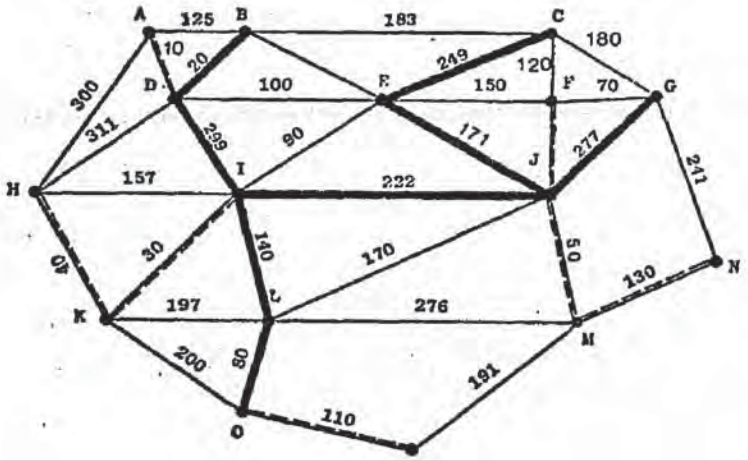


Figure 2.4b. THE OLD SPANNING TREE.



of replacing old branches continues, until all N-1 branches of the old spanning tree have been replaced by new ones. This results in an MST. In figure 2.4c, B, G and O are the newly created leaves after nodes A, F, K, H, M, N and P have replaced the old-branches incident to them. Signals that transfer master control or create a branch contain information about the fragment as seen by the originator of the signal. Note that if a leaf and therefore a master node wishes to make an old-branch into a new one, it need not remove the old one and then create the new one in two separate signals. The node could just create the new one, thereby implicitly removing the old one.

All N nodes in the network will eventually become master, but there are only N-1 branches to be replaced. Some node will be the last one to become master. It will upon examining its fragment state realize that there are no nodes lying outside the fragment, and therefore conclude that the fragment spans all the nodes. Therefore the MST has been constructed. Note that it is not necessary for each node to know the names of all the nodes in the network, or even the total number to make this decision. Hence, new nodes can easily be added to the network. We prove in section 2.6.3 that there is only one node that discovers this fact (unlike the algorithm of section 2.4.3). This node could then broadcast a 'done' signal to all other nodes along the branches of the MST. Upon receiving a 'done' signal a node knows that the last computation phase is over and that it can reinitialize its state information - for example, the new MST now becomes the old spanning tree, and edge costs must be reestablished. Once a node has

reinitialized itself, it can proceed with the next computation phase when it becomes a leaf without having to wait for all other nodes to have got the 'done' signal. We examine how a computation phase terminates and the next one initialized in detail in section 2.6.3.

2.6.2 Statement of the Algorithm

We now describe the algorithm formally. We assume that the only change between two phases is the edge costs changing. We assume that nodes and edges do not come up or go down. The algorithm accounts for these variations, but we leave it for section 2.6.5 to explicitly show this.

2.6.2.1 State Information at Each Node

The statement of the algorithm will assume that each node has a set of state variables. These consist of:

(i) The state of the node with respect to the construction of the new MST; i.e. whether it is inactive, active, master, or done, and the state of the node with respect to the old spanning tree; i.e. whether it is a leaf or not. The state determines what the node should do when it gets a message from another node.

(ii) Information about each of the edges the node is connected to. The information contains source and destination node identities of the edge, its cost, whether the edge is an old-branch or a new-branch, and if a new-branch whether this node and/or the node at the other end made it into a branch

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.