underlying asynchronous mechanism guarantees reliable communication between two nodes.

2.4.3.3 Associated Routines

There are some special routines at each node. MERGE_FRAG_STATE 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. Note that the fragment state gets altered only when a signal arrives and is processed, and not when a signal is produced. Hence, when a node makes an edge into a branch, the fragment state is unaltered as this node does not know what lies beyond the node at the other end of the edge.

A routine called MERGE_EDGE_INFO merges the edge information received in the signal (if any) with that contained for this edge at the node.

DECIDE is a routine that determines which of two nodes should become master. If DECIDE returns true this node 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 fragment states). The node that becomes master could be the one that has a lower cost edge excluding the one that connects both together.

ANY_NEIGHBOR is a routine which examines the fragment state and determines which node (if any) should become master. If ANY_NEIGHBOR returns true, then the identity of this node is returned in MASTER_NODE, and the identity of the node at the other end of the edge from MASTER_NODE in DEST_NODE. The edge determined by (MASTER_NODE, DEST_NODE) connects <u>this</u> fragment to its nearest neighbor. If ANY_NEIGHBOR returns false, then there are no more neighbors of this fragment and thus the MST has been constructed.

TRANSFER_MASTER_CONTROL is a routine that examines the fragment state through ANY_NEIGHBOR. If there is a neighbor, and if MASTER_NODE is the node itself, then the node converts the edge determined by (MASTER_NODE, DEST_NODE) into a branch if it already was not one, and signals the DEST_NODE to 'become master and make this edge an unmarked branch'. If the edge is already a branch then DEST_NODE is signalled to 'become master'. If MASTER_NODE was not the node itself, then MASTER_NODE is signalled to 'become master'.

2.4.3.4 The Main Program

Abstractly, the program in each node can be formulated as follows.

<u>Initialization</u>: Determine the cost of the edges incident at this node and the identities of the nodes at the other end of the edges. Build the data structure corresponding to the edge information and the fragment state. The latter will contain only this node and its edges.

<u>First step</u>: Convert the edge to the nearest neighbor into a marked branch, and signal the neighbor to 'mark this edge'.

<u>General step</u>: Wait for a signal. When it arrives MERGE_FRAG_STATE and MERGE_EDGE_INFO.

If the command is 'mark this edge', then if the edge was marked by both nodes then DECIDE who should become master, as this is a doubly marked edge. If this node becomes master then TRANSFER_MASTER_CONTROL. If the edge was not marked by both nodes do nothing.

If the command is 'become master' then TRANSFER_MASTER_CONTROL.

If the command is 'become master and make this edge an unmarked branch', then if the edge was made into an unmarked branch by both nodes then DECIDE who should become master. If this node becomes master then TRANSFER_MASTER_CONTROL. If both nodes did not make the edge into and unmarked branch then TRANSFER_MASTER_CONTROL.

Repeat the general step.

<u>2.4.4 Analysis of the Algorithm</u>

We put off discussing the factors influencing the complexity of the algorithm till section 2.4.4.2, and now prove that it does in fact construct the MST.

In terms of the general model for parallel MST algorithms (cf. section 2.3 General step), when this algorithm starts, each node is a fragment and converts the edge connecting it to its nearest neighbor into a branch. The neighbor is informed of this by a message. 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.

The arrival of messages determines which fragments have merged into larger fragments, and which nodes are permitted to repeat the general step based on their knowledge of the fragment.

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 (merging fragments) and continues to wait for other messages. 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, since it determines which nodes can repeat the general step (cf. section 2.3). Recall that Marked Fragments are connected together by unmarked branches to create the MST. The master node is, therefore, now 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. Since the MST is unique, so is the branch that connects a fragment to its nearest neighbor, 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. 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 because the edge it picks is not the lowest cost edge, and its inclusion as a branch 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 are always marked, 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.

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.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.