Summary of "A Distributed Algorithm for Minimum-Weight Spanning Trees"

Janne Lindqvist

April 11, 2005

Abstract

This document summarizes the article published by Gallagerher et. al on "A Distributed Algorithm for Minimum-Weight Spanning Trees". The asynchronous distributed algorithm determines a minimum-weight spanning tree for an undirected graph that has distinct finite weights for every edge.

1 Introduction

This document summarizes a classical article by Gallagher, Humblet and Spira published in 1983 [1]. In addition for the summary, I elaborate on some topics that the authors of the original article considered as preliminaries.

The rest of the paper is organized as follows. In Section 2, I give the preliminaries and fundamental assumptions for understanding the algorithm. Next, in Section 3, I present the distributed algorithm. Brief summary of the analysis of the algorithm is given in Section 4. Finally, in Section 5, I conclude the paper.

2 Preliminaries

An undirected graph is defined as $G = (V,E)$, where V is a finite nonempty set and $E \subseteq V \times V$. The V is a set of nodes v and the E is a set of edges e. The graph is *connected* if there is a path between any distinct e. A graph $G_s = (V_s,E_s)$ is a *spanning subgraph* of $G = (V,E)$ if $V_s = V$. A *spanning tree* of a graph is an undirected connected acyclic spanning subgraph. Intuitively, a spanning tree for a graph is a subgraph that has the minimum number of edges for maintaining connectivity. [2]

Let $w(e)$ be a *weight* for an edge *e* in a graph. Then, the weight of a tree is the sum of all the $w(e)$ in the tree. A *minimum-weight spanning tree* (MST) is a spanning tree where the sum of $w(e)$ is minimal.

A *fragment* of an MST is a subtree of the MST. An *outgoing edge* is an edge of a fragment if there is a node connected to the edge in the fragment and one node connected that is not in the fragment.

A *level* of a fragment is defined as follows. A fragment with a single node has the level L = 0. Additionally, "suppose a given fragment F is at level ≥ 0 and the fragment F' at the other end of F's minimum-weight outgoing edge is at level L'. If $L < L'$, then fragment F is immediately absorbed as part of fragment F'." The expanded fragment is at level L'. "If L = L' and fragments F and F' have the same minimum-weight outgoing edge, then the fragments combine immediately into a new fragment at level L+1; the combining edge is then called the core of the new fragment." Otherwise, "fragment F waits until fragment F' reaches a high enough level for combination under the above rules".

The two essential properties of MSTs for the algorithm in the article are:

> **Property 1** *Given a fragment of an MST, let e be a minimum-weight outgoing edge of the fragment. Then joining e and its adjacent nonfragment node to the fragment yields another fragment of an MST.*
>
> **Property 2** *If all the edges of a connected graph have different weights, then the MST is unique.*

The proofs of the properties are given in the article. Next, I will give the underlying assumptions for the distributed algorithm.

2.1 Assumptions

The authors state the following assumptions for the algorithm:

- each node initially knows the weight of each adjacent edge
- each node performs the same local algorithm
- messages can be transmitted independently in both directions of an edge
- the messages arrive after an unpredictable but finite delay
- the messages contain no errors
- the messages arrive in sequence
- the weights for each edge are **distinct** and finite

Next, I describe how the distributed algorithm works under the stated assumptions.

3 Description of the Distributed Algorithm

The algorithm definies three different states of operation for a node. The states are *Sleeping*, *Find* and *Found*. The states affect what of the following seven messages are sent and how to react to the messages. The messages are **Initiate**, **Test**, **Reject**, **Accept**, **Report(W)**, **Connect(L)** and **Change-core**. The *identifier* of a fragment is the *core edge*, that is, the edge that connected the two fragments together.

When a single node wakes from the Sleep state, it processes the following procedure:

Procedure 1 *Node awakens*
Select the adjacent edge with minimum weight
Mark the selected edge as *Branch*
Send Connect(L) over the edge
State = Found
Wait for response from the other end of the edge

The above procedure was the simplest case. Assume a fragment that consists of more than one node. The fragment is level L and has been combined of two fragments of level (L - 1). The combination of the fragment was established by a single minimum-weight edge, which is called the core edge of the new fragment. As explained above, the core edge is now the new identity of the fragment.

Two nodes adjancent to the core edge broadcast the Initiate message. The Initiate message is propagated to all of the nodes in the fragment. The Initiate messages consists of the fragment level, the identity and Find flag as parameter. Thus, the Initiate message puts the receiving nodes in the Find status and informs them about the new identity and fragment level. Additionally, the Initiate message is passed to fragments at level (L - 1), so that they can join in the new fragment.

When a node receives the Initiate message, it processes the following procedure:

Procedure 2 *Classify adjancent edges*
if edge = branch **then**
  classify as **Branch**
end if
if edge is not branch but joins two nodes of the fragment **then**
  classify as **Rejected**
end if
if edge is not branch and not rejected **then**
  classify as **Basic**
end if

After the above procure, the node sends the Test message to the minimum weight **Basic** edge. The Test message contains the fragment identity and level

– Test(level). If the Test message is responded with the Reject message, the node sends the Test message to the next best edge. If the Test message is responded with the Accept message, the node knows that the edge is an outgoing edge from the node's fragment. The Test messages are processed by recipients as follows:

Procedure 3 *Test message processing*
if fragment identity is same as mine **then**
  send Reject
else
  if if my fragment level $\geq$ Test(level) **then**
    send Accept
  end if
end if

By following the above procedures, all nodes in a fragment eventually find existing minimum-weight outgoing edges. Each *leaf* node now sends the Report(W) message on its inbound branch. The W is the weight of the minimum-weight outgoing edge or infinity if the node has no outgoing edges. If no node has outgoing edges, the algorithm is **complete** and the fragment is the minimum-weight spanning tree.

Each *interior* node of the fragment wait until it has received the Report(W) messages from all outbound fragment branches and found its own minimum-weight outgoing edge. The node compares its own minimum-weight outgoing edge and Report(W) messages and marks as *best-edge* the edge that has the smallest weight. Then, it sends the Report(W) messages on its inbound branch and transitions to the Found state. The best-edge marking is used as a backward route for the Change-core message to be described.

The Report(W) messages are propagated in the network and eventually reach the core edge. The nodes adjancent to the core edge can then determine on which side the new minimum-weight outgoing edge is. The Change-core message is now sent to the path marked with best-edge. The Change-core message changes the inbound edge for each of the nodes traversed to correspond to best-edge. When the Change-core message reaches the node with the minimum-weight outgoing edge, the node sends the Connect(L) message over the edge. The L is the level of the fragment.

The Connect(L) message has three possible outcomes. First, consider two fragments that have the same minimum-weight outgoing edge and the fragments are on the same level L. The fragments now combine into a new fragment of level (L+1) and the minimum-weight outgoing edge is designated as the new core edge. The new fragment sends the Initiate message described earlier and thus begins the procedures again. Second, consider that the node which sends the Connect(L)

message belongs to a lower level fragment than the recipient. If the recipient node has not send a Report(W) message, the two fragments join. Then, the new fragment tries to find a minimum-weight outgoing edge. Third, if the recipient node has send a Report(W) message, the fragments join but the fragment with the lower level does not participate in finding the new minimum-weight outgoing edge.

The authors also outline a proof for the algorithm and give a short explanation of a version that does not require distinct weights but requires unique and ordered identities for the nodes.

4 Analysis of the Algorithm

The authors of the article show that the upper bound for the number of messages exchanged during the execution of the algorithm is $5N\log_2+2E$, where N is the number of nodes and E is the number of edges in the graph. A message contains at most one edge weigth and $\log_2 8N$ bits. A worst case time for the algorithm is $O(N\log N)$.

5 Conclusion

The paper presented an asynchronous distributed algorithm that determines a minimum-weight spanning tree for an undirected graph that has distinct finite weights for every edge. As such, the algorithm is not usable in real networks because of the assumption of distinct weights. A modified version is however in use in modern devices that connect local area networks (LAN). The devices are called *bridges* and they form a spanning tree for ensuring that broadcast messages are delivered properly. A more elegant version of the algorithm is used in another type of bridged LANs, in systems called virtual local area networks (VLAN). Distributed spanning tree algorithms are also used by multicast routing protocols for establishing connectivity to only the needed nodes (routers) in the network.

References

[1] GALLAGHER, R. G., HUMBLET, P. A., AND SPIRA, P. M. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems 5*, 1 (Jan. 1983), 66–77.

[2] GRIMALDI, R. P. *Discrete and Combinatorial Mathematics, An Applied Introduction.* Addison Wesley Longman, Inc., 1999.