Distributed systems - Course Project

Introduction

We code the Gallager Humblet Spira (GHS) algorithm from scratch in this project. GHS is an algorithm to construct Minimal Spanning Trees of graphs in a distributed setting. To do so, we hold a major assumption: every edge in the graph has unique weights and that, if every edge in a graph is unique, then the MST is unique.

Idea

The idea of GHS (briefly) is as follows:

- A **fragment** is a part of MST. The goal of the algorithms is to form fragments, join them, and terminate only when one fragment remains. The borders of the fragment are made up of nodes that have one outgoing edge inside the fragment and another outgoing edge towards another fragment. A single node can also be a fragment.
- Initially, each node is a fragment. Fragments fuse together to make larger fragments. Nodes in a fragment run a distributed algorithm to locate the overall minimal outgoing edge. On consensus or a general set of rules, fragments combine with the core edge being the new minimally outgoing edge.

- Each fragment has a unique name. Each fragment has a level. When two fragments combine, one of them changes their name and their levels stay the same / get updated depending on a set of rules.

Rules

Rules for the combination of two fragments

Let the combining fragments be F_1 and F_2. Let their levels be L_1 and L_2 respectively. F_1 can only combine with F_2 only if $L_1 < L_2$. If $L_1 = L_2$, the fragments can combine when they reach a consensus on if the outgoing edge chosen by one of them is minimal for the other fragment too.

Rules for a level update on the combination of fragments

If $L_1 < L_2$, All nodes in the new fragment have names F_2 and level L_2. If $L_1 = L_2$ and the outgoing edges $e_{F1} = e_{F2}$ for both the fragments, the new level of the combined fragment is $L_1 + 1$ and the new name of the fragment is e_{F1}. If these rules don't apply, wait till these rules apply hoping that one fragment eventually gets back.

Algorithm

Variable Name	Type	**Values Inherited**	**Description**
State	Integer	sleep, find, found	sleep: Node is not initialized. find: Node is currently in search of a minimal outgoing edge. found: The minimal outgoing edge has been found.
Status	Array	each element of the array inherits basic, branch, reject	For a given node p, status stores the array of its edges. basic: Edge is unused. branch: Edge is a part of the MST. reject: Edge is not a part of the MST.
name	Integer	any edge weight	Name of the fragment.
level	Integer	any number between 0 and	Level of the fragment.

		$log_2 k$ where k is the total number of nodes.	
parent	Integer	Node indexing.	Parent of a given node in MST, pointing t

- **Initialization.** Each fragment's core node sends a branch request to its children, and the children send it to their neighbors. Since at this point, a fragment is a single node, find the minimally outgoing edge and get ready to send a connection request.
- **When a connection request is received from a node, q.** Request branch from the current fragment, initiate a branch request to children, and propagate only if the level of q is lesser than that of the current fragment. If the level is greater, then wait. Else, the levels are similar and in order to reach consensus, search for the smallest outgoing edge by searching and propagating
- **On receiving Initiate from q.** Initialize temporary variables to store the best outgoing edge and the best outgoing weight. Send initiate messages to children, search for minimal outgoing edges for the given node, wait for responses from children and decide.
- **For finding the minimal outgoing edge.** To find the minimal outgoing edge, send a **test** request to another fragment with which the most minimal outgoing edge is present. If no response is gained from the test, then terminate the search.
- **On receiving** a **text message from q.** If the level of the current fragment in which the associated node is present is lower than the q's level, then wait. Reject if both the nodes belong to the same fragment. Else, **find the minimal outgoing edge.** If no reply or the minimum outgoing edge is the edge associated with q, **accept.**
- **On receiving accept from q.** Update best weight and best node parameters, propagate back to parent using **report.**
- **On receiving report from q.** Update the status of the current edge from basic to reject.
- **On receiving report from q.** Change status to found, and report to the parent. If the current node is the root node of the fragment, and if the status is find, then wait. If the weight of the edge from report is greater that the best outgoing edge the root has found, then implement the **change root** function.

- **Change root.** Change the root of the nodes to the best node, propagate to children.

Functions

Analysis

At any level, a node receives the following:

- 1 initiate message.
- 1 accept message.

At any level, a node sends the following:

- 1 report message.
- 1 change root message / 1 connect message.
- 1 successful test message.

Every node is rejected only once and hence, over a single edge: 1 test message, 1 reject message.

Message Complexity: $2E + 5Nlog(N)$

Implementation

The code for the algorithm can be found here:

The code has been written using C++ and we have used Open MPI for node message passing.