**Distributed Computing Group Project Report**

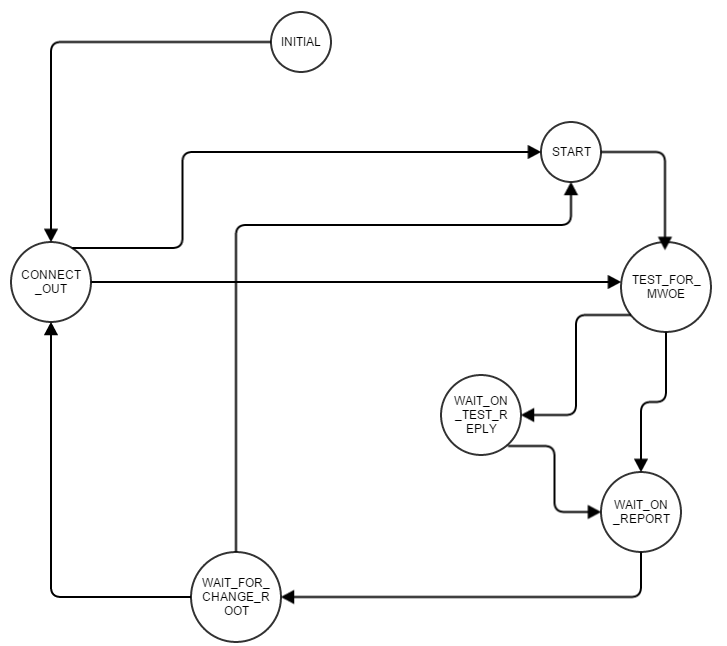
**MST using Asynch GHS Algorithm**

**By**

**Geoffrey Grimmer**

**Siddharth Perumal**

System Design:



We approached the design by starting with the Asynch messaging structure. We used a 2D shared memory matrix for message passing, in which the index of the row is the sender of a message while the column is the receiver of the message. Every entry in the 2D matrix is a C++ list that acts as the FIFO for message queueing between two particular processes.

We implemented a set of states that correspond to the message types that we have sent out and the message types that we can currently expecting.

Apart from this, the “Test” and “Connect” messages can be handled in any of these states as and when they are received. Differing of test messages is still handled based on the component levels.

To achieve this we have used “IF” blocks, where each thread goes to state and executes only the block of code belonging to its state.

Each process (thread) starts at the “INITIAL” state with its level as 0. It then post a connect a message to its nearest neighbor and moves to the “CONNECT\_OUT” state. Then upon successful connect it moves to start state to receive an initiate or if it is leader, it posts the initiate message.

A leader, that posts an initiate message, moves directly to the TEST\_FOR\_MWOE state. The processes in the START state will also move to TEST\_FOR\_MWOE upon receiving an initiate message. These process, use the information in the initiate message to update their level and component information. They also update their parent, because only a parent process can send an initiate.

Once in the TEST\_FOR\_MWOE state, processes will send a test message to their edge with smallest weight. This achieved by sorting the list of neighbors by their edge weights and breaking ties with smallest process ID to prevent deadlock. If a process has no more basic edges, it skips to the WAIT\_FOR\_REPORTS step. Otherwise, it sends a test message to the neighbor with smallest edge weight. Processes then move to WAIT\_FOR\_TEST\_REPLY.

In the WAIT\_FOR\_TEST\_REPLY state, the process checks its messages for an accept or a reject message. Upon receiving a reject we move back to TEST\_FOR\_MWOE. If we receive an accept message we move to the WAIT\_FOR\_REPORTS step.

While in the WAIT\_FOR\_REPORTS step, processes will check how many non-parental branches are connected to them. This is the number of report messages they must receive before processing them and finding a local MWOE. This local MWOE is then reported to the parent. If a process in this state is the leader of their component, they do not send a report message, instead broadcasting a change root message. Processes, who send a report message, move to the next state and wait for a change root message.

Upon receiving a change root message, the process whose ID matches the process ID stored in the message will send a connect message along the MWOE and move to the CONNECT\_OUT state. All other processes move to the START state to wait for initiation.

A process, in the CONNECT\_OUT state, wait and evaluate whether or not a merge or absorb is occurring. If a process receives a connect message back along the same edge, we know it is a merge operation. A connect message along a different edge, is another component trying to be absorbed into my component. A process, will know it has been absorbed into a larger component if an initiation message comes in along the MWOE instead of a connect message. In this case, the process will update parent to the sender of the initiate and forward to all other branches incident on the process. In the case of a merge operations, the large ID becomes the new leader and broadcasts an initiate, while the process with a smaller ID marks the other process as a parent. Theses move directly to TEST\_FOR\_MWOE.

When the process in the CONNECT\_OUT state forward or broadcast the initiate messages, it signals for processes in the START state to search for an MWOE. This process continues until a point where a component leader has received reports from all of its neighbors that indicate they could not find an MWOE and the component leader cannot find an MWOE. This signals the termination of the algorithm and all the threads terminate.