Practically comparing Tarrys algorithm and Awerbuchs

Shawn Biesan
School of Electrical and
Computer Engineering
Kent State University
Kent, Ohio 30332–0250
Email: sbiesan@kent.edu

***Abstract*—Tarrys and Awebuch's algorithms are two wave algorithms that are used for spanning tree construction. I show a comparison of their time complexities in graphs of various number of nodes and densities. Based on the performance results I explain under which conditions each algorithm should be used.**

I. Introduction

This paper will present a performance comparison of Tarry's [3] and Awerbuch's [2] algorithms. It is organized as follows: first the algorithms and method of comparing performance will be introduced, then the next section will explain how the experiments are set up, then explanation of the simulation code, the results, and finally concluding thoughts and possible future work.

II. Background

Some background on the algorithms to be compared. Both algorithms are waves, meaning that they terminate, decide, and each decide event is causally preceded by an action of each process. They are used for spanning tree construction. Time complexity will be used to measure the performance of each algorithm. Time complexity is defined as the longest chain of causally related messages in an algorithm.

Tarry's algorithm is a transversal algorithm, meaning it is a wave and there is only one initiator and one message is sent at a time. Tarrys algorithm works as follows: The initiator forwards token to one of its neighbors, each neighbor then forwards token to all other nodes and when done returns the token. Its theoretical time complexity is 2 * number of edges in the graph. [1]. More information about the algorithm can be found in the original paper [3].

Awerbuch's algorithm is a wave but not a transversal algorithm since a process can send multiple messages. It works as such: a node notifies neighbors that it is visited by sending a VIS message so tokens are never sent over frond edges. Its theoretical time complexity is 4 * number of nodes in the graph - 2. It should be noted that the time complexity of Awerbuch's algorithm is based on the number of nodes in the graph and Tarry's algorithm is based on the number of edges of the graph. [1]. More information can be found in the original paper [2]

III. Experiment Setup

Two experiments are to be performed. The experiments will compare the time complexities of the two algorithms. Each data point will be the result of averaging 5 trials.

The first experiment will compare the time complexity of the algorithms as the number of nodes varies. They will vary from 5-50 by increments of 5. The topology will be fixed as fully connected graph. Since this is a fully connected graph its number of edges is $\frac{(N*N-1)}{2}$ where N is the number of nodes in the graph. In Theory, Awebuch's algorithm should significantly outperform Tarry's as N increases since its time complexity scales linearly in terms of N and Tarry's is quadratic.

The second experiment will be varying the density of a partially connected graph and comparing their time complexities. A paramater p will be used to vary this and it will correspond to the probability that an edge exists between two nodes. P will vary from 30%-100% by increments of 10%. The number of nodes will be fixed at 10 for this experiment. The generated graph must be connected so if it is not, then the graph is regenerated. Tarry's algorithm should outperform Awerbuch's for low p and at some point Awerbuch's should overtake it as the graph becomes closer to fully connected.

IV. Code

The experiment and simulation engine is coded on C++. Algorithms are ran in the simulation engine by implementing a specific interface that assures that an algorithm has methods that can do things such as check if any guards are enabled, receive a message, and execute a single command of the algorithm. There are also expected variables algorithms need in this interface such as a message queue and datastructure to store a nodes neighbor. Each node is actually a specific implementation of this class. In theory this code could have nodes of different algorithms in the same computation, but that is not needed.

When generating a partially connected graph, the code checks that the graph is connected by running depth-first-search on the graph and making certain every node can be reached. The code supports arbitrary additions of topologies and algorithms.

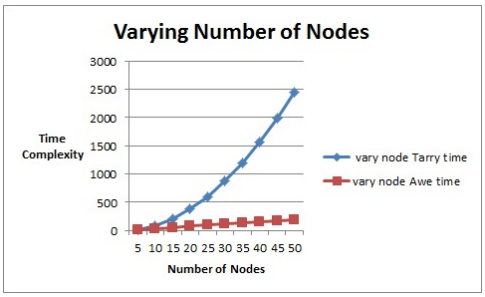

Fig. 1. Varying nodes 5-50 by increments of 5

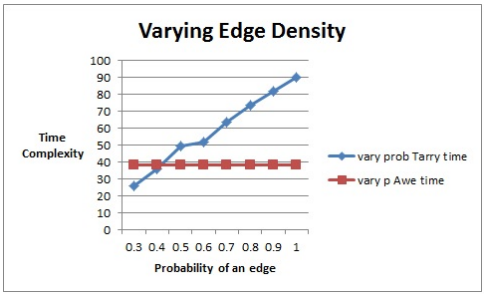

Fig. 2. Varying probability of an edge 30%-100% by increments of 10

V. Results

Based on the first figure as nodes scale in a fully connected graph, Awerbuch's is indeed better. This is directly based on their theoretical time complexities. Awerbuch's scales linearly with respect to number of nodes, whereas Tarry's scales quadratically.

Based on the second graph Tarry's is indeed better for sparse graphs. This trend continues up until about p = 40%. The expected number of edges in a graph with p = 40% is 18 so this coincides with what would be theoretically expected.

VI. Future Work and Conclusions

In general Tarry's algorithm should be used for sparse graphs with small number of nodes, otherwise use Awerbuch's. It is easier to implement and there isn't an overhead of VIS, ACK messages. In the future I would like to explore these algorithms on different topologies such as various trees, stars, chains, and others. I would also like to see if there are interesting properties related to message complexity of these algorithms.

References

[1] M. Nesterenko, *Slides of advanced operating systems class*, http://deneb.cs.kent.edu/ mikhail/classes/aos.f07/.

[2] B. Awerbuch, *A new distributed depth-first search algorithm*, Inform. Process. Lett. 20 (1985) 147-150.

[3] G. Tarry, *Le Problem Des Labyrinthes*, Nouvelles Annales de Mathematique 14, 1895.