Empirical Analysis of Randomized Leader Elections

K. Nathaniel Tucker
Harvard University
tucker@college.harvard.edu

ABSTRACT

Electing a leader in a distributed system, or the equivalent task of nominating a unique node, is a fundamental problem in distributed computer systems. In the implicit version, only the leader or nominee need know that she was selected. All other nodes must know that they were not elected, but need not know the identity of the leader. This paper will summarize previous work done on this problem, provide intuition behind the results achieved, in some cases provide more practical bounds, and chiefly provide an empirical analysis of the performance of the most efficient to date leader election mechanisms. Performance will be measured mainly across two vectors: time complexity and message complexity, but we will also provide other statistics when necessary.

General Terms

Computer Communication Networks, Distributed Systems, Graph Theory, Leader Election

1. MOTIVATION

With applications ranging from the emerging area of large scale and resource constrained networks (eg. peer to peer networks) [11] to large ad hoc and sensor networks [3], the study of the leader election in distributed computing is as relevant today as ever. By minimizing key components like messages and time, efficient algorithms for leader elections can help minimize energy consumption in real world systems in sensor networks.

2. INTRODUCTION

The leader election problem requires a group of agents in a distributed network to elect a unique leader among themselves. At the end of the process, the goal is to have one and only one agent to change the special status in its state to $leader$, while all other agents change their status to $non-leader$. The version of leader election that we will focus on is *implicit* leader election. In this version, the other nodes need not know the identity of the leader node. This *implicit* version of leader election is standard [9] and is generally sufficient for many applications. In the *explicit* version of leader election, every node knows the identity of the leader, and while the algorithms I describe do not focus on the explicit version, they can easily be extended to solve this problem.

The study of leader election (both explicit and implicit) concerns itself primarily with two measurements: message and time complexity. For deterministic algorithms under some strict conditions we can make some claims. The conditions are: (a) the algorithms are only comparison algorithms (as in they cannot manipulate nodes identities, but rather only compare them), (b) spontaneous wakeup of nodes is not guaranteed, and (c) network parameters (like the number of nodes) are not known. In this case a lower bound of $\Omega(m)$ messages where m is the number of edges holds [13].

It was only recently that bounds were proven for universal leader election algorithms, or algorithms that work for all graphs. These bounds are $\Omega(m)$ messages and $\Omega(D)$ time (where D is the diameter of the graph) [7]. Both of these bounds subsume the deterministic bounds and act as new randomized bounds. They apply to a large class of graphs, for nearly all m, D, n. They hold for non-comparison algorithms. They hold for synchronous networks, even if all the nodes wake up simultaneously. And they hold for not only the $CONGEST$ [12] framework where sending a message of $O(\log n)$ bits takes one unit time, but also for the $LOCAL$ framework where the number of bits in a message is arbitrary.

Note that these bound only hold for universal leader election algorithms, meaning that we could and there have been algorithms developed for particular graphs with better runtimes. Kutten et al. showed a randomized algorithm that uses $O(\sqrt{n}\log^{3/2} n)$ messages for a complete graph for example [8].

And when it comes to randomized algorithms, one needs to be careful. Consider this Monte Carlo algorithm: Each node elects itself with probability $1/n$. Then the probability that exactly one leader is elected is:

$$\binom{n}{1}\frac{1}{n}(1-\frac{1}{n})^{n-1} \approx 1/e \approx 0.368$$

Therefore for any graph, I have stated an algorithm that runs in $O(1)$ time with $O(1)$ messages. Thus the above bounds require the success probability to be a bit larger. I will provide intuition, in some cases re-derive a different bound, and in others outline these proofs later in the paper.

Given these lower bounds, a natural question stems: Is it possible to design a universal algorithm for leader election that is simultaneously both time and message optimal [12]? Or in light of the previously given lower bounds: is it possible to design a $O(D)$ time and $O(m)$ messages universal leader election algorithm? Deterministically the answer is no. It is known that for a cycle any $O(n)$ time deterministic algorithm requires at least $\Omega(n \log n)$ messages (even when n is known) [2]. However this problem is still relevant for randomized algorithms.

It was recently shown that there exists a randomized algorithm that matches both complexities for $m \geq n^{1+\epsilon}$ for any fixed ϵ assuming n is known. And one that matches both complexities with n and D known [7].

However, in other papers, we have seen algorithms that seem to defy these lower bounds, like the solution for the complete graph [8]. Remember that the results are for universal leader election algorithms, and if an algorithm is designed for a particular graph, it can have better runtimes. Therefore this paper will seek to explore the relationship between graph type and leader election performance in the empirical results section.

This paper comes in four parts: (1) providing background necessary to understand the proofs, algorithms, and empirical experiments later in the paper, (2) provide simplified versions and intuition for the lower bound proofs, (3) step though the implementation details of four algorithms that meet these lower bounds, and (4) describe my empirical experiments of some of these algorithms in practice.

I will end with results, conclusions and future work.

3. BACKGROUND

3.1 Leader Election

We consider an undirected connected graph $G = (V, E)$. Each node v will run an instance of our distributed algorithm, the leader election. Each node has a unique identifier ID_v (sometimes called rank) of $O(\log n)$ bits chosen uniformly over an arbitrary set of integers Z where $|Z| = n^4$. The nodes themselves may not have knowledge of n nor Z. And even so, some of the algorithms in the paper do not require knowledge of the D itself. For example the randomized algorithms in this paper will also apply for anonymous networks. However for all implementations, I assume knowledge of D, M, n.

We assume the best case for node synchronicity, and all nodes wake up simultaneously. This is also assumed for implementations.

Time steps forward in synchronous rounds, where each round nodes can send messages, receive messages sent from neighbors, and preform some local computation.

For all randomized algorithms we assume that each node has access to the outcome of private unbiased coin flips, however, they cannot access the coin flips of other nodes and do not share any memory.

Initially each node is given a series of ports, where each port is connected to an edge leading to a neighbor. However, the node has no knowledge of the neighbors at the endpoint of the edge.

In the leader election itself, every node v has a special variable $status_v$ that can be set to a value in $\{\perp, non{-}elected, elected\}$. Initially $status_v$ is set to $\perp$. An algorithm solves a leader election in T rounds if, T is the minimum round such that onwards, exactly one node has status set to *elected* and all other nodes are in state $non-elected$.

We call L a universal leader election algorithm with error probability ϵ, if for any choice of $n = |V|$ and $m = |E|$ the probability which L solves a leader election on any graph $G = (V, E)$ is at least $1-\epsilon$. This holds for any ID assignment chosen from the integer Z such that $Z \geq n^4$. Moreover, if L a deterministic algorithm or a randomized Las Vegas algorithm, then L is universal if and only if it achieves leader election on ever graph under all ID assignments.

3.2 Network Topology

Here we will give a definition of what network topology is, and a justification for studying it in relation to leader elections.

Network topology is loosely speaking the arrangement of various elements of a network. And depending on the way that nodes are distributed on a graph, such a graph will have different topologies.

Our paper will consider an undirected graph $G = (E, V)$. And there will be a set of features that are important for us to understand about the graph. We will let d_{ij} be the shortest distance between nodes $v_i, v_j \in V$. We define n to be the number of nodes in V. We define the neighbors of a specific node as $N_i = \{v_j : e_{ij} \in E\}$, and thus we define $|N_i|$ to be the degree of $v_i \in V$. Finally we define the clustering of a specific node as:

$$c_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}}{(|N_i| - 1)|N_i|} \tag{3.2.1}$$

These definitions are necessary for defining the two aspects of graph topology that will interest us: characteristic path length and clustering. The combination of the two will be referred to as having Small World properties. These will be studied on constraint networks.

We will define these properties as:

1. Characteristic path length of $L(G)$

$$L = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \tag{3.2.2}$$

2. Clustering coefficient $C(G)$

$$C = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{3.2.3}$$

Finally we will define high clustering and low characteristic path length to be having the Small World property. We will now show why we wanted to study the Small World property vs. the many others.

3.2.1 *Natural Network Topologies*

In Jin & Liu 2003, they studied naturally occurring networks and the topologies thereof. They discovered that in many network problems, the networks could have the Small World topology, depending on how it was formulated. If each node in the system was controlled by one agent, then the Small World property was observed. If instead the agents were organically laid out over the variables, the Small World property was not. The conclusion was that different topologies can occur via different constructions. [4]

This ability to easily change the topology of the network, allows designers to further optimize the process by first composing the network into the most favorable topology. This shows us why the Small World property, out of many others should be studied.

Moreover in Araujo & Lamb 2008, they discuss creating networks that are optimal for various types of problems. Thus where the designers are able to actually construct the network itself, knowledge of where algorithms succeed would be very important. [1]

In sum, the Small World property is generally inherent to real social networks, and can be easily constructed and removed artificially into and from a network. Thus this paper will explore the Small World property to see which approximate algorithm preforms best with and without it.

3.2.2 *Small World*

The Small World property itself is actually quite important. Many natural and man-made systems exist in the form of networks, and research has found that many such networks exhibit a these characteristics in their topology, which means that the shortest distance between any two nodes is quite small and that the nodes are highly clustered.

These properties bring about efficiency for a network. In Jin & Liu 2003, they argued that a network with these two properties would support efficient communication both globally and locally. They defined ϵ_{ij} to be the communication efficiency between to vertices v_i, v_j. They suggest $\epsilon_{ij} = \frac{1}{d_{ij}}$ where d_{ij} is the shortest path between the two. Based on this they defined the communication efficiency of the entire network G to be as follows:

$$E_{globe} = E(G) = \frac{1}{n(n-1)}\sum_{i\neq j\in G}\epsilon_{ij} = \frac{1}{n(n-1)}\sum_{i\neq j\in G}\frac{1}{d_{ij}} \tag{3.2.4}$$

Then assuming that G_i is a sub-network of G composed only of the neighbors of v_i (including itself) and the edges between them. Then the local communication efficiency of G is as follows:

$$E_{loc} = \frac{1}{N}\sum_{i\in G} E(G_i) \tag{3.2.5}$$

It was found that E_{globe} would be high in networks with the low characteristic path length, and that E_{loc} would be high in networks with high clustering. Thus one would predict that networks with high communication would be able to solve global problems more easily. [5]

However such a prediction may often turn to be the opposite. Walsh 1999 proved that such a topology: high clustering and low characteristic path length, can make search problems very difficult. The search cost actually increased as the graph's characteristic path length decreases. [14]

Thus the crux of our project is to show empirically where each claim holds.

4. LOWER BOUNDS

Before we jump into describing the algorithms and the experiment design, we should begin by understanding the bounds that control our system.

There are two lower bounds for universal leader election algorithms, including Monte Carlo algorithms with a suitably large constant success probability: a message lower bound of $\Omega(m)$ on the number of messages, and a time bound of $\Omega(D)$ on the number of steps it will take till completion of the algorithm.

For the message complexity bound we show that for every m, n there exists a graph of with $\Theta(n)$ nodes and $\Theta(m)$ edges where the lower bound holds. As for the time lower bound we show that for every n and $D \in (2, n)$, there exists a graph with $\Theta(n)$ nodes and $\Theta(D)$ diameter for which the time needed is $\Omega(D)$, even with constant success probability. This bound holds even if each node had knowledge of the global parameters n, D, m. The bound holds for all algorithms and even if the nodes wake up simultaneously.

4.1 Message Lower Bound

The message lower bound is proved first by showing a lower bound for a related problem called bridge crossing BC. In the bridge crossing problem, it is required that at least one message is sent across a bridge edge connecting two specific sub-graphs. Then by creating a graph such that any universal leader election on this graph must solve BC. This proves the lower bound on deterministic algorithms [7]. Then one can use Yao's Lemma to imply the bound carries over to randomized algorithms of on worst-case input [10].

The proof lends itself to more detail, but I will give the broad strokes.

4.1.1 *Construction of Dumbbell Graphs*

The first part of the proof begins with construction of a family of graphs which we will preform bridge crossing algorithm on. We will use $k-$connected graphs with n nodes and m edges, that I will call G, in our construction. Given one such graph G', I designate the removal of edge e' from the graph (leaving the ports empty) as $G'[e']$.

From this I will construct a dumbbell graph, DB. To construct this graph, I take two such graphs $G'[e']$ and $G''[e'']$, and I connect the ports of the two graphs. This will give me a new graph that I will label as $DB(G'[e'], G''[e''])$.

Then for every two graphs G' and G'' I define the class $C(G', G'')$ to be the set of all DB graphs over G' and G''. The size of $C(G', G'')$ is m^2.

We let I be the set of all possible dumbbell graphs. The size of I is $m^2|C(G', G'')|$. And the number of total classes is the number of graphs squared $|G|^2$ (any graph on the left paired with any on the right).

4.1.2 Bridge Crossing Complexity

Now that we have constructed our graphs, I will show that for every deterministic algorithm A, and for each graph G' and G'', if A achieves bridge crossing on at least ϵm^2 of the graphs in the class $C(G', G'')$, for some constant $0 < \epsilon \leq 1$, then the expected message complexity of A on inputs taken from $C(G', G'')$ with a uniform distribution is $\epsilon^2 m/8 = \Omega(m)$

The proof begins with an experiment. Preform A on two disconnected copies of G', and denote it as $EX(G')$. Then for each edge e determine the first time $t(e)$ that a message was sent over e. Then run a similar experiment on G'', $EX(G'')$.

The critical observation is that the $EX(G')$ is identical to the execution on $DB(G'[e'], G''[e''])$, up to the point that a message crosses e' or e''! Therefore if e' was first crossed, then $t(e')$ would correspond to the time in which a message first crossed the bridge from G' to G'' in $EX(DB(G'[e'], G''[e'']))$.

Now let us consider the problem again. Consider the m^2 executions on $C(G', G'')$. In ϵm^2 this algorithm succeeds. And without loss of generality, in at least $\epsilon m^2/2$ of these executions the first crossing message was sent from G' to G''.

Let me then define C_i to be the subset of all graphs in $C(G', G'')$, such that the $t(e') = i$ (that is the time of the first crossing over the left bridge is i). We know that the size of all C_i's is at least $\epsilon m^2/2$ by the argument made above, or $\sum_{i=1}^{k} |C_i| \geq \epsilon m^2/2$. We further know that the size $|C_i|$ represents the number of executions of $EX(DB(G'[e'], G''[e'']))$ where the first message was sent from G' to G'' over edge e'_i. And because there are exactly m DB graphs in $DB(G'[e'_j], G''[e''])$, we know that $|C_i| \leq m$.

Now let Q be the total number of messages sent, which we can lower bound by $Q \geq \sum_{i=1}^{k} i|C_i|$, where k must be greater than $\epsilon m^2/m$ (minimum number of blocks of size m to cover ϵm^2 terms). Therefore:

$$Q \geq \sum_{i=1}^{\epsilon m} i|C_i| \geq m\frac{\epsilon m(\epsilon m+1)}{2} \geq \epsilon^2 m^3/4$$

There are a total of m^2 graphs in $C(G', G'')$, and we have shown a total lower bound on messages in these graphs to be $\epsilon^2 m^3/2$, therefore the expected cost incurred by A over the class $C(G', G'')$ is at least $\epsilon^2 m/8 = \Omega(m)$.

4.1.3 Reduction to Leader Election

In the second step of the proof, I will show if a deterministic algorithm A solves leader election on a fraction of the input, then A must achieve bridge crossing on a fraction of the output as well. To do this, we will show that there is no algorithm that can achieve a high success probability of leader election without having both sides of the DB communicate with each other, or in other words bridge crossing.

We will show for $\epsilon \geq 0$, $\delta \geq 0$, and $2\sqrt{\epsilon} + \epsilon + \delta \leq 1$, if A achieves leader election on at least a $1 - \epsilon$ fraction of the input I, then A achieves at least δ fraction of the graphs in I.

Let $I_{LE} \subseteq I$ be the set of graphs that A solves leader election on. We know that $|I_{LE}| \geq (1 - \epsilon)|I|$. Let $I_{BC} \subseteq I$ be the set of graphs that A solves BC on. Assume towards contradiction that $|I_{BC}| < \delta|I|$. Let $I^* = I_{LE}/I_{BC}$. Then we know that:

$$|I^*| > (1 - \epsilon - \delta)|I|$$

Now let L be the set of graphs $G'[e']$ that participate on the left in the set of DB I^*. We can lower bound the size of L by:

$$(1 - \epsilon - \delta)|G|^2 m^2 = (1 - \epsilon - \delta)|I| < |I^*| \leq |L||G|m$$

Or in other words, If we take each graph on the left and combine it with any possible graph ($|G|$) via any possible edge on that graph (m), then that must be greater than or equal to the size of I^*.

Now we move on to the key observation. Consider an execution of A on the DB, $DB(G'[e'], G''[e''])$. The graph $G'[e']$ either reached the state with all nodes in the state $non-elected$, or with only one node in the state $elected$. And for all other DB in I^* such that $G'[e']$ participates, regardless of which side, it will reach the same result as before (because there is no communication with the other graph!). This means that graphs that participate on the left side in I^* are also the graphs that participate on the right (call them R), because the reverse DB graph $DB(G''[e''], G'[e'])$ must also succeed if $DB(G'[e'], G''[e''])$ succeeded. Therefore we can partition the graphs L (and similarly R) into two sets $L_{NE} = R_{NE}$ (the graphs that end with all nodes in state $non-elected$) and $L_E = R_E$ (where one node ends $elected$). The key observation is that for every pair of open graphs from L_{NE} and R_{NE}, A will fail the LE, because neither side elected a leader.

So without loss of generality, assume that $L_{NE} > |L|/2$. We can use this to lower bound the number of DB graphs X that algorithm A fails to solve LE on. By our key observation, this must be greater than or equal to $|L_{NE}||R_{NE}|$. Then by our assumption this is greater than $|L|^2/4$. Then by the above:

$$X \geq |L_{NE}||R_{NE}| > |L|^2/4$$

$$> (1-\epsilon-\delta)^2|G|^2m^2/4 = (1-\epsilon-\delta)^2|I|/4$$

Then by our original assumption we know that $X \leq \epsilon|I|$. Therefore we can say that $\epsilon|I| > (1-\epsilon-\delta)^2|I|/4$. Then by rearranging we get: $2\sqrt{\epsilon}+\epsilon+\delta > 1$, which contradicts our original assumption.

4.1.4 Proof

I have shown you that any deterministic algorithm that solves LE for a fixed fraction of the inputs must solve BC for a fixed fraction of the inputs, and I have shown you that any deterministic algorithm that solves BC for a fixed fraction of the inputs has an expected message complexity of at least $\Omega(m)$ on the uniform distribution of those inputs.

The final step of the proof is the application of Yao's Minmax Principal that states: For a finite distribution of inputs Φ, let X be the minimum expected cost of any deterministic algorithm that succeeds on at least a $1-\beta$ fraction of inputs. Then $X/2$ lower bounds the expected cost of any randomized algorithm on the worst case graph in the inputs that succeeds with probability $1-\beta$. The application of Yao's Principal completes the proof.

There are better bounds in other papers, but above are the broad strokes.

4.2 Time Lower Bound

The time lower bound is significantly simpler, and is proven directly for Monte Carlo algorithms (instead of using Yao's Lemma) by a probabilistic argument. The intuition behind the bound, is that diameter represents the distance needed to travel such that all nodes can communicate with each other, and if there are some nodes that do not communicate, then we cannot achieve a high probability of success.

4.2.1 Graph Construction

The graph that we are going to construct this time is a cycle. There are four sections of the cycle C_1, C_2, C_3, C_4. C_1 is connected to C_2 which is connected to C_3 which is connected to C_4 which in turn is connected to C_1. The distance between C_1 and C_3 as well as C_2 and C_4 is $\Omega(D)$ which is the diameter of the graph. And finally, the layout of each C_i is the same as all the others. In other words, each C_i is isomorphic. We call the entire graph C.

4.2.2 Proof

Consider a universal leader election algorithm R that succeeds with probability $1-\beta$. Then there exists a graph G with $\Theta(n)$ nodes and $\Theta(D)$ diameter where R takes $\Omega(D)$ rounds with constant probability.

First consider the random variable T denoting the running time of the assumed algorithm R. Assume towards contradiction that the event $T \in o(D)$ happens with constant probability $\delta = 1-o(1)$.

Next let L_i be the event where that there is one leader elected in C_i. And correspondingly let $L_{i,j}$ be the event that there is only one leader elected in C_i and C_j. I claim that for all i, j, $P[L_i|T \in o(D)] = P[L_j|T \in o(D)]$. We can see this via induction. For the first round, all $v \in C_i$ observe the same neighborhood as the $u \in C_j$, therefore we have the event being equally likely. Thus by inductively applying this symmetry argument $P[L_i|T \in o(D)] = P[L_j|T \in o(D)]$. I will then call $P[L_i|T \in o(D)] = p$ for the sake of convenience.

We can also easily see, for C_1 and C_3 as well as C_2 and C_4, if $T \in o(D)$, then the events L_1 and L_3 (respectively L_2 and L_4) are independent. Therefore we have $P[L_{1,3}|T \in o(D)] = 2p(1-p)$ (and the same for $L_{2,4}$).

Therefore we have:

$$P[L_{1,2,3,4}|T \in o(D)] = \frac{P[L_{1,2,3,4}]}{P[T \in o(D)]} - \frac{P[L_{1,2,3,4}|T \in \Omega(D)]P[T \in \Omega(D)]}{P[T \in o(D)]}$$

$$= 1 - \frac{\beta}{\delta}$$

Remembering that $P[L_{1,2,3,4}] = 1-\beta$ and $P[T \in o(D)] = \delta$. We can also bound this term with:

$$P[L_{1,2,3,4}|T \in o(D)] \leq P[L_{1,3}|T \in o(D)] + P[L_{2,4}|T \in o(D)] = 4p(1-p)$$

Or in words, the probability that one leader is elected, must be less than the probability that a leader is elected on the left plus the probability that it is elected on the right. This gives us $p \geq (1-\sqrt{\beta/\delta})/2$. But because we know that $\beta \geq p^2$ (the probability that there is an error must at least be greater than the probability that two leaders are elected), we get: $(1-\sqrt{\beta/\delta})^2/4 \leq \beta$ which requires $\delta < \sqrt{\beta}$. Contradiction. Therefore $T \in \Omega(D)$ with constant probability.

5. ALGORITHMS

This section is primarily about achieving the lower bounds though the design of algorithms. We present four algorithms below that seek to achieve this bound through tradeoffs of success probability and prior knowledge about the system.

The first two algorithms that we will present are both deterministic. First is a deterministic universal algorithm that achieves the message lower bound of $O(m)$ messages, however takes an arbitrary time that depends on the size of the smallest ID [2].

The second universal algorithm achieves the time lower bound $O(D)$, using the least elements lists described below.

This algorithm uses the least element lists (LEL) [6]. It can be used to elect a unique leader with probability 1 in $O(D)$ time and $O(m \min(\log n, D))$ messages. The main idea behind the algorithm is to use unique IDs. Each node simply propagates its unique ID and the largest ID wins. If the IDs are chosen randomly, then it can be shown that the number of messages each node v has to forward is bounded by $O(logn)deg(v)$ where $deg(v)$ is the degree of v.

The next two improve this with a randomized approach. One can improve this bound by restricting the number of nodes that can be candidates. This achieves the lower bounds $O(m)$ messages and $O(D)$ time complexity by trading off success probability (albeit maintaining a large constant success probability).

Then we finally show that if we have knowledge of n, D then we can obtain a Las Vegas algorithm with expected time complexity $O(D)$ and expected message complexity of $O(m)$.

5.1 Message Lower Bound Algorithm

The MLB Algorithm is best described as a convergence of wandering agents [2]. Each node initiates an annexing agent that does a DFS over the entire graph. The agent leaves its unique ID at each node she visits. And if the agent ever encounters a node that has a lower ID than itself, the agent halts.

The rule that allows a reduction of messages is that an agent whose ID is i preforms one DFS step each 2^i steps. The algorithm is better described below:

Algorithm 1 MLB: Message Lower Bound Algorithm

```
 1: Initialize each node, $v_i \in V$ to unique ID $id$
 2: Activate each agent $A_i$ for each $v_i \in V$ with unique id
    $id$, and $Q_i = [v_i]$
 3: for Each Activation Step $s$ do
 4:     for Each Agent $A_i$ do
 5:         if $s\%A_i.id! = 0$ then
 6:             Continue
 7:         end if
 8:         if $Q[1].id < A_i.id$ then
 9:             Kill agent
10:         end if
11:         $A_i.visited+ = [Q[1]]$
12:         $Q[1].id \leftarrow A_i.id$
13:         $Q[1].status \leftarrow non-elected$
14:         for Neighbor $n_i$ in $Q[1]$ do
15:             if $n_i \notin A_i.visited$ then
16:                 $Q+ = [n_i]$
17:             end if
18:         end for
19:         $Q.pop()$
20:     end for
21: end for
```

Notice that in the algorithm above, as soon as an agent finds a node with a lower ID than hers, she immediately stops her traversal. Also notice that if a node's ID is ever overwritten, that node can immediately assume the state $non-elected$. There is a node with a lower ID, therefore it cannot become the leader.

The agent with the lowest ID will completely traverse the graph with a total of m steps. The agent with the second lowest ID can at most traverse $m/2$ edges. By this time the agent with the lowest ID will have also traversed m edges, and the next edge that the second agent must pass through has already been traversed by the first agent. This will stop the second agent. The third agent can then only traverse $m/4$ edges before the first agent traverses all the nodes. Therefore the message complexity is bounded by $O(m + m/2 + m/4...) = O(m)$.

In terms of implementation, the details are rather straightforward. I did ensure that a node with ID zero was in the system, and even so you will see an explosion of steps.

5.2 Time Lower Bound Algorithm

The next algorithm that we describe uses the least element. The data structure assumes that each node has a unique ID. Let $DIST(u, v)$ be the length of the shortest path from u to v. Then the least element list for a particular node u is $LE_u = [u, v_1, ..., v_t]$, such that each $v_i \in LE_u$ is the element with the lowest ID in the $DIST(u, v_i)$-neighborhood of u. Notice that this list can only have a length of the diameter of the graph, and that v can show up in the list multiple times (so long as it is adjacent to itself).

For the randomized least element list algorithm [6], each node chooses its rank randomly, and forwards its rank to its neighbors. Upon receiving u's message, the neighbor v then adds u to its LE_v if u's rank is strictly smaller than the last element in LE_v, and otherwise discards it. During the next round, if and node u added an element to its LE_u, then it forwards that rank to all its neighbors.

Notice that all nodes receive all possible messages from distance r in round r, and thusly can only add one entry per round. It follows that no new entries are added to LE_u after D rounds therefore the time complexity of this algorithm is $O(D)$. It is shown that $|LE_u| \in O(\log n)$ for every node which implies the total number of messages sent during the algorithm is $O(m \log n)$ [6] .

The algorithm succeeds given that the minimum rank is unique. The node with the lowest ID should be added to all nodes LE_u and that node should have no other elements in its LE_v. Therefore the node that has no elements in its least elements list calls itself leader after D time steps. I have formalized the algorithm below:

In terms of implementation, I made this a deterministic algorithm. I ensure that all ranks were unique. In addition I reduced the size of the message passed along, by allowing each node to send back to the agent that sent the message. In such a way, messages are only propagated if they are

Algorithm 2 LEF: Least Element Full Algorithm

```
1: Initialize each node, $v_i \in V$ to unique id $id$
2: Initialize each nodes, $v_i.LE$ to unique id $[id]$
3: For each $v_i \in V$ broadcast its unique id to all neighbors
4: for Each Activation Step do
5:     for Each node $v_i$ do
6:         Let $\min(v_j.id)$ be the minimum id received via broadcast last round
7:         if $v_i.LE.last > \min(v_j.id)$ then
8:             $v_i.LE += [\min(v_j.id)]$
9:             Broadcast $\min(v_j.id)$ to all neighbors
10:        end if
11:    end for
12: end for
```

lower than the rank of the current node. I assumed that nodes had knowledge of D, n, m as well, though the above could be implemented without such knowledge.

This algorithm yields very good complexity bounds, but we can do better. In our next algorithm we show that we can achieve the lower bounds in expectation if we allow for a constant probability of failure.

5.3 Least Element Monte Carlo Algorithm and Las Vegas

In this algorithm, we use the same procedure as above, but we only allow a select few nodes to be candidates. In this algorithm each node u becomes a candidate with probability $f(n)/n$, for some fixed $f(n) \leq n$ where $f(n) \in \Omega(1)$. And in such a case the expected size of the LE_u for each node is bounded by $O(\min(\log f(n), D))$.

To see this, consider a node v. Let N_k be an ordering of the candidates within the k-neighborhood of v, such that N_k is non-decreasing with respect to distance to v. We know that the expected size of N_k is on the order of $O(f(n))$ (which is the expected number of candidates). Now consider the LE_v, however let us construct a least elements list that can contain multiple entries of the same distance. This will always be greater than the original LE_v. We can construct this new list, call it LA_v, by looking at N_k and taking from N_k all elements such that the ID of element i is less than the ID of all previous elements. Because the IDs are uniformly distributed, $E[LA_v] = \sum_{i=1}^{N_k} i^{-1} = O(\log f(n))$.

Because of this if we choose $f(n) = 4\log(1/\epsilon)$, we will get a message complexity of $O(m)$ and a success probability of at least $1 - \epsilon$. This algorithm uses the same pseudo-code as above with one difference. That is, each node only becomes a candidate with $4\log(1/\epsilon)/n$ probability.

We can easily take this an make it into a Las Vegas algorithm as well.

In this algorithm, if we fail, we simply rerun the algorithm. Because there is a constant probability of success, the expected number of reruns is constant as well and we can achieve an expected message complexity of $O(m)$, time complexity of $O(D)$ and success probability of 1.

Both of these algorithms can be implemented with only modifications to the above, and it was rather straightforward.

6. EXPERIMENTAL DESIGN

We will be designing and using: the above algorithms and a network generator (particularly important is the Watts-Strogatz model) for this experiment.

6.1 Small World Topologies

There are two properties that we will model over our topologies:

1. Characteristic path length of $L(G)$: 3.2.2

2. Clustering coefficient $C(G)$: 3.2.3

The characteristic path length measures the typical separation between two vertices. The average clustering coefficients measures cliquishness of a typical neighborhood on the graph. The paper will be using the Watts-Strogatz model to achieve both.

6.1.1 Watts-Strogatz

The Watts-Strogatz model utilizes two graph structures: a ring lattice and an Erdos-Renyi random graph. The ring lattice has high clustering coefficients, and the Erdos-Renyi has low shortest path.

The Watts-Strogatz model then interpolates the two. Starting with a regular ring lattice, the model rewires each edge at random with some probability p, where if $p = 0$ it will remain a regular ring lattice, and if $p = 1$, it will become an Erdos-Renyi.

The paper will examine behaviors over this model with probabilities ranging from .01 to .1. We will then overlay a leader election atop these created graphs.

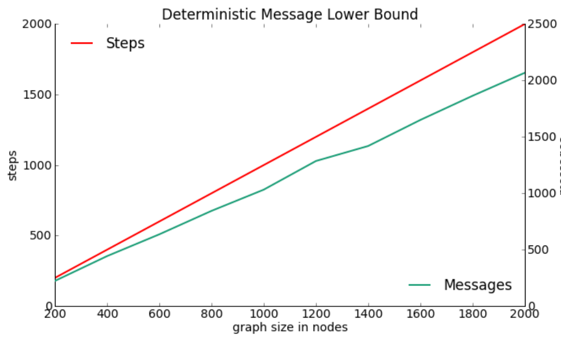
6.2 Evaluation of Algorithms

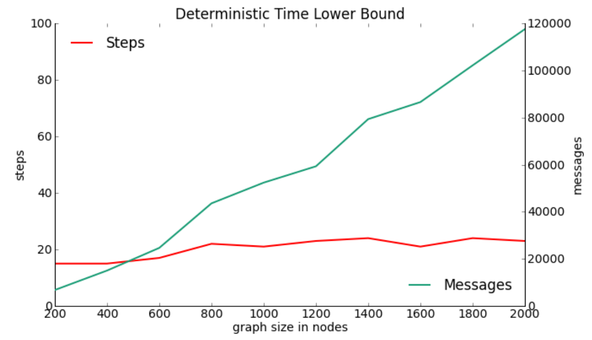
The algorithms will be evaluated on three major criteria. Each will be run for n trials where $n = 100$.

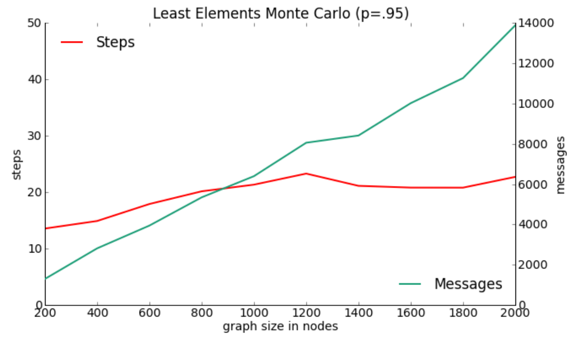
The algorithms will then be evaluated on message complexity, time complexity and success probability. Message complexity being the average messages sent over the 100 trials (including failures). Time complexity being the number of steps taken before a leader is elected averaged over the 100 trials (again including failures). And for the Monte Carlo algorithm we will keep track of the success probability.

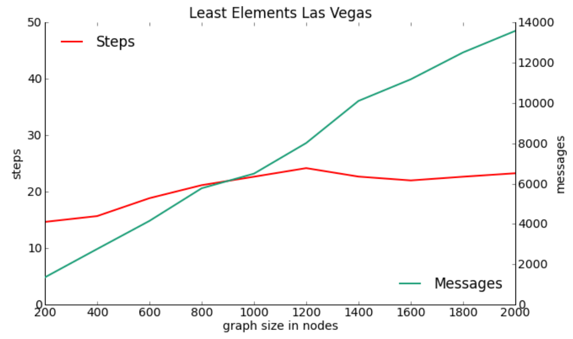
7. RESULTS

The following results are empirical, however the paper will try to provide some intuition as to why one sees these results. Then the paper will suggest a set of guidelines for algorithmists to use given an underlying knowledge of the network in terms of characteristic path length and clustering coefficients. This can be further used to show the best results given that one can choose topology and algorithm.









7.1 Vary the Size of the Graph

In the first section we vary the size of the graph to see how tight the bounds are empirically

We ran all four of the algorithms on the same Watts-Strogatz generated graphs for 100 trials. The graphs ranged from 200 to 2000 nodes each with degree 5 (lager graphs will be tried in the future). We used a $p = .05$ for the Watts Strogatz parameter. And for the ϵ bound on probability failure we used .5.

As you notice, the most surprising thing is that the bound of failure probability of .5 gave us a 95 percent success rate for the Least Elements Monte Carlo approach, and did not significantly raise the number of steps or the message complexity of the Las Vegas version. In both of the Least Elements approaches we see a linear increase in number of messages.

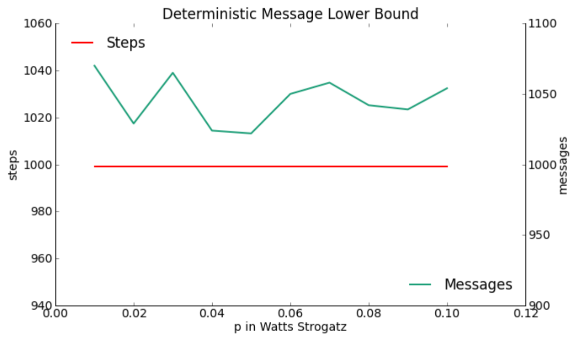
As for the Time Lower Bound approach, we see similar results for steps the algorithm takes. Similar to the Least Elements approaches. The message complexity is significantly greater (nearly 10 times so). We do not however see a polylogarithmic effect with this number of samples (though there was little change in the diameter).

As for the Message Lower Bound approach, we see another constant factor cut in the number of total messages (around 7 times). But we see a near 100x increase in number of steps. This being so even through I used the optimal ID allocation (IDs ranging from 0 to $n - 1$). But we can easily see the tradeoff.

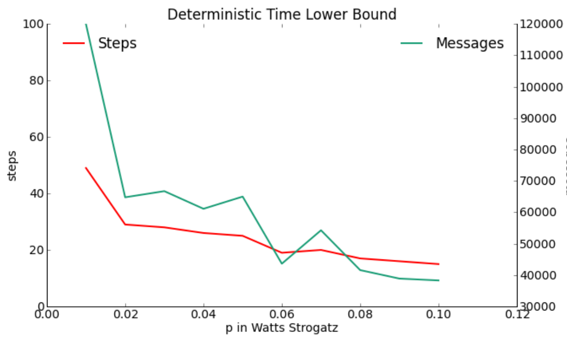
I think that this gives us intuition for a clear superiority of the Las Vegas approach (given that we know n and D). Empirically it preformed nearly as well as the Monte Carlo yet always succeeded. One could imagine a situation where the cost of sending a message is very expensive, and in that case the Message Lower Bound might be a better option. The most surprising fact is that the ϵ lower bound is not very tight for regular graphs. However the most interesting results are next.

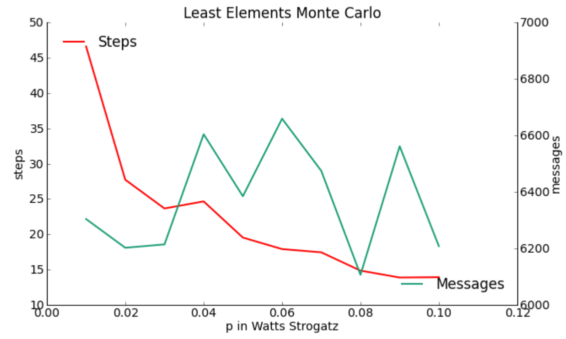
7.2 Vary the Parameter of Watts Strogatz

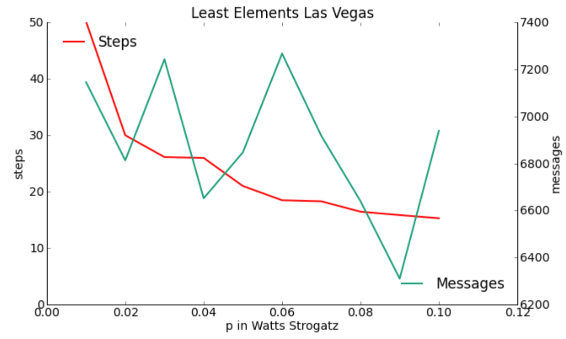
The second experiment we preformed was to vary the topology of the graphs. Therefore we varied the p in the Watts Strogatz model. We varied the p of the Watts-Strogatz model from .01 to .1 in increments of .01. In such a way we were able to vary the small world property and the clustering coefficients of the graphs. The results are below.



What should not be odd about any of these graphs, is that as p becomes closer to 1, the number of steps decreases. Do note that there are fluctuations in the number of messages sent for both the least elements approaches (we will get to that later).





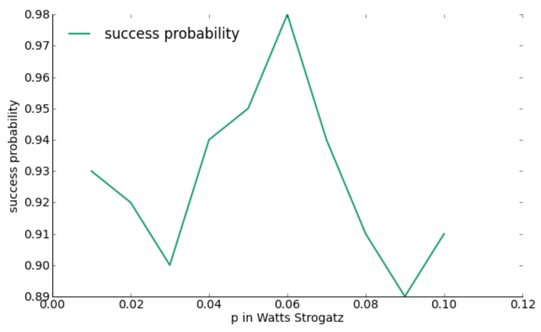


Perhaps the most interesting result is that the Time Lower Bound approach also has decreasing message complexity as the network becomes less clustered and has a smaller diameter. This should naturally happen because the message complexity in the Time Lower bound algorithm depends on the diameter. But we saw the message complexity go from 20x to 5x by decreasing the diameter.

Perhaps the most interesting result is the probability of success for the Least Elements Monte Carlo approach (below).

This is perhaps the most interesting of all. While the variations are not too large, we see a peaked probability of success around a p of .06, or the probability when we have both high clustering and smaller diameter. This is perhaps the most interesting result of the paper, and we will discuss it more during the conclusion.

8. CONCLUSION



In conclusion, we can see that the empirical results match the theoretical. The Time and Message lower bounds seemed to follow linear relationships in D and m. The Time Lower Bound's message complexity also observed a logarithmic decrease when we changed the diameter of the network.

Interestingly the Message Lower Bound saw a 10x decrease in number of messages as compared to the Least Elements approach that achieves the same lower bound.

Probably most interestingly, we saw variations of the success probability when we varied the Small World characteristics of the network for the Least Elements Monte Carlo approach. This might also account for the odd fluctuations in message complexity of the Las Vegas approach.

For the most part the theory matched what was happening in practice.

9. FUTURE WORK

The most interesting bit of future work will revolve around new theory for probability over different network topologies and new empirical results.

However much work can be done. More trials with wider variation in n and p can and should be done. More algorithms should be studied in regards to performance over various network topologies. There are many things that can be done.

10. APPENDIX

For code used to produce results (as well code for two other leader election algorithms not seen in this paper) please see https://github.com /knathanieltucker/leader$_e lection$

11. REFERENCES

[1] R. M. Araujo and L. C. Lamb. Memetic networks: Analyzing the effects of network properties in multi-agent performance. In *AAAI*, pages 3–8, 2008.
[2] G. N. Frederickson and N. A. Lynch. Electing a leader in a synchronous ring. *Journal of the ACM (JACM)*, 34(1):98–115, 1987.
[3] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149. ACM, 2003.

[4] X. Jin and J. Liu. Agent network topology and complexity. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 1020–1021. ACM, 2003.

[5] X. Jin and J. Liu. Efficiency of emergent constraint satisfaction in small-world and random agent networks. In *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pages 304–310. IEEE, 2003.

[6] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012.

[7] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. On the complexity of universal leader election. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 100–109. ACM, 2013.

[8] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. Sublinear bounds for randomized leader election. In *Distributed Computing and Networking*, pages 348–362. Springer, 2013.

[9] N. A. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.

[10] R. Motwani and P. Raghavan. *Randomized algorithms.* Chapman & Hall/CRC, 2010.

[11] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[12] D. Peleg. Time-optimal leader election in general networks. *Journal of parallel and distributed computing*, 8(1):96–99, 1990.

[13] G. Tel. *Introduction to distributed algorithms.* Cambridge university press, 2000.

[14] T. Walsh et al. Search in a small world. In *IJCAI*, volume 99, pages 1172–1177, 1999.

[15] D. J. Watts and S. H. Strogatz. Collective dynamics of âĂŸsmall-worldâĂŹnetworks. *nature*, 393(6684):440–442, 1998.

[16] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM Sigmod Record*, 31(3):9–18, 2002.