ELECTION AND TRAVERSAL IN UNIDIRECTIONAL NETWORKS

Eli Gafni
Yehuda Afek

March 1985
Report No. CSD-850008

Election and Traversal in Unidirectional Networks

Eli Gafni
Yehuda Afek

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

This paper presents distributed algorithms for election and traversal in strongly connected unidirectional networks. A unidirectional network consists of nodes which are processors connected by unidirectional communication links. Initially, processors differ by their identifier but are otherwise similar. The election algorithm distinguishes a single processor from all other processors. The election algorithm requires $O(\log n)$ bits of memory in each processor and has communication complexity of $O(n \cdot m + n^2 \log n)$ bits. In the traversal algorithm one node initiates a token which visits all the nodes of the network and returns to the initiator. The traversal algorithm is derived from the election algorithm. It achieves the same communication complexity and uses only $O(1)$ bits of memory in each processor.

1 Introduction

We consider a strongly connected unidirectional network, that is, a strongly connected directed graph in which each node is a processor and each arc is a unidirectional communication link from the processor at the tail of the arc to the processor at its head. We study this model since it is the most general network on which fundamental graph algorithms can be carried out distributively.

We present two algorithms: election of a leader, and traversal. The election algorithm is an identical program which resides in every node in the network. Thus, all nodes execute the same program. Initially, nodes differ only by their identifier (*id*). The nodes exchange messages, and when the message exchange ends, a single node in the network has distinguished itself as a leader. The message exchange may be initiated by an arbitrary set of nodes. The election algorithm yields two spanning trees, both rooted at the leader; one is an incoming tree, and the other is an outgoing tree. In the traversal algorithm, which we assume to be started by a single node, a token visits all the nodes of the network and returns to the

*This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-82-C-0064. The second author is an IBM Graduate Fellow.

initiating node.

Election algorithms have been previously proposed in the context of bidirectional networks and unidirectional rings. Gallager, Humblet and Spira [Gall83] proposed an $O(m+n \log n)$-*messages* algorithm for bidirectional networks, where n is the number of nodes and m is the number of arcs in the network. Burns [Burn80] showed that this complexity is a lower bound. Peterson [Pete82] and Dolev et al. [Dole82] proposed an $O(n \log n)$-*messages* algorithm for a unidirectional ring. Pachl et al. [Pach82] and Fredrickson and Lynch [Fred83] proved that this is a lower bound. Recently, Gafni and Korfhage [Gafn84b] proposed an $O(m \log n)$-*messages* election algorithm for a unidirectional Eulerian graph.

The election algorithm presented in this paper has a complexity of $O(n \cdot m + n^2 \cdot \log n)$-*bits* for general unidirectional networks, using $O(\log n)$ *bits* of memory in each node. The algorithm is an improvement on the connectivity checking algorithm of Segall [Sega83] and the shortest path algorithm of Gallager [Gall76], that can be easily modified to work on a unidirectional network. The communication complexity of the two algorithms is $O(n \cdot m \log n)$ bits, and each node is assumed to have $O(n \log n)$ bits memory. Furthermore, unlike our algorithm, neither Segall's nor Gallager's algorithm provides the spanning trees.

The traversal algorithm simulates depth first search and requires a constant amount of memory in each node. The algorithm is a special case of the election algorithm, in which only a single node starts the algorithm. It incurs the same communication cost as the election algorithm. The significance of the traversal algorithm is that it provides a means of conducting fundamental operations, (e.g., majority vote) on a unidirectional network of finite automata, on condition that a single automaton initiates the algorithm. Similar issues are studied in [Koze79, Shan71].

The paper is organized as follows: Section 2 describes the network model; the election algorithm is described in section 3, and its complexity is discussed in section 4; the traversal algorithm is presented in Section 5; and concluding remarks are given in Section 6.

2 The Network Model

Our model follows the traditional message passing model [Burn80, Lync81] with two exceptions. First, the underlying communication network is unidirectional, and second, the set of operations on the input and output ports is more restrictive. Each processor has a set of input ports and a set of output ports. Communication between processors is established by connecting an output port of one processor to an input port of the other. Messages going over such connections incur an unpredictable, but finite, delay and arrive at the input port in the order sent. Strong connectivity of the underlying network is the only restriction placed on port interconnection.

Ports of a processor can be marked with a finite set of marks. Ports are referred to only by their marks. Initially, connected output ports of a processor are distinguished from all other output ports by a special mark. The connected input ports of a processor are initially indistinguishable from the unconnected ports. A port mark may be changed only upon transmission or reception of a message.

For the election problem, each processor has a unique *id* of $O(\log n)$ bits and $O(\log n)$ bits of scratch-pad memory. In the traversal algorithm, each node is a finite automaton, i.e., the size of the scratch pad is constant, and processors do not have unique ids (i.e., the way we define local memory will agree with the standard terminology when all nodes are of bounded degree).

The communication complexity of our algorithms is expressed in terms of the total number of bits transmitted over the communication links. This is different from the usual approach, in which the total number of messages of size $O(\log n)$ is counted.

3 A Unidirectional Election Algorithm

3.1 Definitions and Out Line

The algorithm is based on the following properties of strongly connected directed multigraphs:

1. The set of arcs, defined by selecting one outgoing arc from every node, contains a nonempty set of disjoint directed cycles.

2. The subgraph, obtained from G by contracting any of the cycles defined above into one node, results in a strongly connected multigraph.

3. Repeated application of the operations in 1 and 2 contract G into a single node.

The distributed algorithm proceeds in conceptual phases, which follow the above contraction process. When a cycle is detected, its nodes are grouped into a cluster. Similar phases are used in [Humb83]. Initially we consider each node to be a single node cluster. The phases of the algorithm are: selection of an outgoing link from each cluster, called a *selected* link; detection of cycles among clusters; and contraction of cycles of clusters.

A cluster is recursively defined as follows:

1. A single node is a cluster.

2. A set of clusters that are joined in a ring by their selected outgoing links is a cluster.

In every cluster, one node is designated as the cluster-head. The *id* of the cluster-head node is also the *id* of the cluster.

During the algorithm, links are in one of three states: *new*, *elementary* or *killed*. A *new* outgoing link is one which has not yet been traversed. An *elementary* link is a link which was a cluster selected outgoing link during one of the previous stages. The set of elementary links within one cluster forms a strongly connected subnetwork, called the *infrastructure* of the cluster. A *killed* link is a nonelementary link already traversed during the algorithm (i.e., an intracluster nonelementary link).

To select a cluster outgoing link, each cluster initiates a Depth First Search Traversal (DFT) process, seeking a link which is potentially outgoing from the cluster. To detect a cycle, we use a simple algorithm for election on a unidirectional ring. Each cluster forwards the largest *id* it has seen. When a cluster receives the same *id* twice it has detected a cycle. The contraction of a cycle is accomplished by first electing one of the cluster-heads on the cycle to be the cluster-head of the expanded cluster. Then, the newly elected cluster-head synchronizes the cluster by broadcasting the new cluster-id to all the nodes in the new cluster.

The most costly phase is the Depth First Search for a cluster outgoing link. The reason being that, in the contraction phase, we lose the DFT information accumulated by all the clusters around the cycle except one. When the cluster-head initiates a DFT in the next cycle, the search will have to spend much effort regaining all the lost DFT information. However, by selecting the cluster-head of the largest cluster on the ring to be the new cluster-head, we minimize the amount of information lost. Thus, we limit the rate of information loss to the rate of cluster growth. (i.e. if a large cluster were contracted with a small one, the amount of information lost is proportional to the size of the small one). In fact, we are able to show that the cost of all the DFT's conducted during the algorithm, is within a constant factor of the cost of a single DFT. Since this point is critical in the complexity calculation, but rather minor to the description of the algorithm, we postpone a detailed discussion of it until the complexity section.

After a cluster is formed, its nodes are synchronized to search for a new link outgoing from the cluster. To achieve this synchronization, inductively an incoming tree rooted at the cluster-head is defined in every cluster. Each node in a cluster, except the cluster-head, has one outgoing link marked as TREE link. The collection of TREE links forms a directed incoming tree, spanning the cluster and rooted at the cluster-head. When a cycle of clusters is contracted into a bigger cluster, all their intrees are merged into one intree, spanning the new cluster. The operations of merging intrees and searching clusters utilize each other alternately. The structures left by the DFT's are used to modify and merge the separate intrees around the cycle into one intree. The intree, in turn, is used for routing purposes, by the DFT process of each cluster. Thus, in the next subsection, where we describe the search method which is used to select a cluster outgoing link, we assume the existence of an intree.

3.2 Distributed Depth First Traversal of Unidirectional Networks

A building block of the election algorithm is the distributed Depth First Traversal (DFT) of unidirectional networks in which an intree is defined. The root node of the intree initiates the DFT by spawning a process which visits all the nodes in the network. Upon arriving at a node for the first time, the process marks the node and recursively spawns one new child process, which sequentially visits each of the node's outgoing neighbors. If a process arrives at an already marked node, it backtracks to its father. After its child process has backtracked from all the outgoing neighbors, it is killed, and the process backtracks to its father. The traversal terminates when the child process of the root node is killed.

To perform the backtracking in a unidirectional network, we use the given intree and note two observations. Nodes on which live processes are located form a simple directed path. In the sequel we call this path an *active path* and its links *active links*. The active outgoing link of each process leads to its child process. The first node on the *active path* is the root of the intree. All backtrackings are from the last node on the *active path* to its father. Thus, to accomplish backtracking, a process follows the unique path of the predefined intree, from its location to the root. From the root, the process follows the *active path* to its father (whose identity each process remembers).

Each backtracking performed requires at most $2n$ messages of $\log n$ bits each. Since there are m backtrackings, the total communication cost of the DFT is $O(n \cdot m \log n)$ bits.

Note that at any given time, all but one of the live processes are waiting for their child processes to terminate. The last process, which has no child and is actively visiting nodes, is called the FOCAL POINT of the DFT. The FOCAL POINT is analogous to the stack pointer in the centralized Depth First Search algorithm.

We can view the DFT as a token traversal scheme in which the token location is the FOCAL POINT of the DFT. In the algorithm each cluster will employ a DFT to choose one outgoing link as the cluster's selected outgoing link.

3.3 Selection of a Cluster-Outgoing Link

In the next three subsections we present the three phases of the algorithm starting with the cluster outgoing link selection (see figure 3.1).

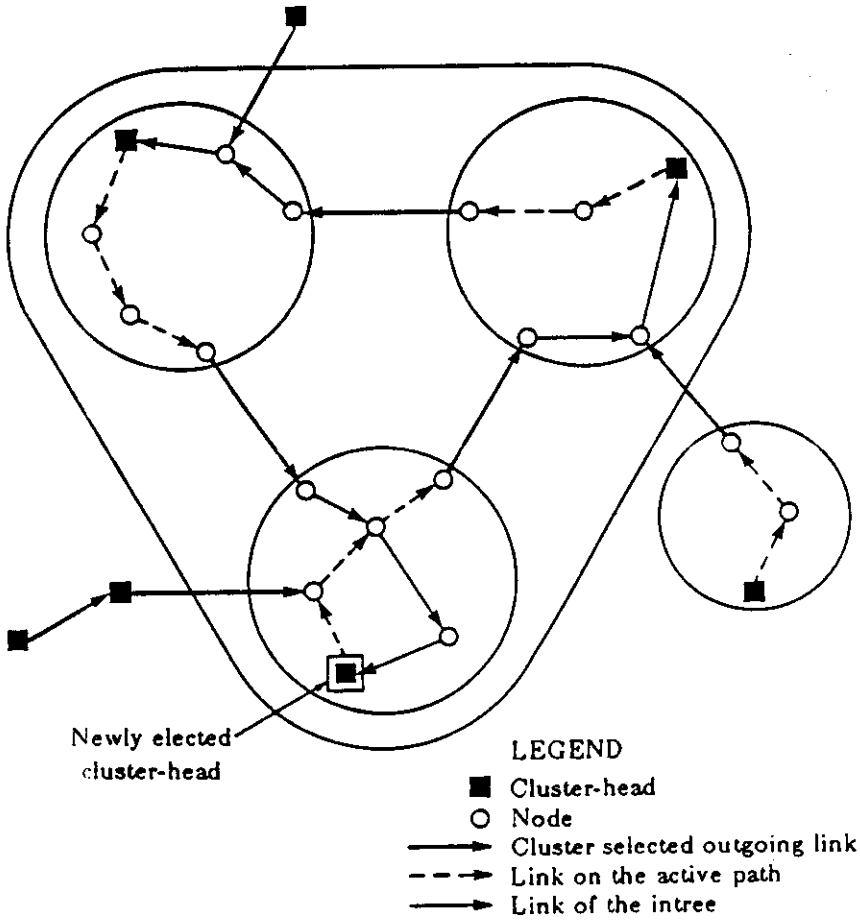

Figure 3.1: Clusters in the Election Algorithm

Inductively, we assume that an in-coming spanning tree rooted at the cluster-head is defined. Initially, this is obviously true.

Once a cluster is formed, its head node initiates a DFT algorithm to search the cluster's *infrastructure* for a node with a *new* outgoing link. The first such link to be found is selected as the cluster's outgoing link. If it turns out to be an intracluster link, another *new* outgoing link is sought. If no cluster outgoing link is found, the cluster contains all the nodes of the network, and the algorithm terminates.

The cluster outgoing link selection phase begins after the synchronization of the cluster has terminated. The head node of the new cluster initiates a DFT token on the cluster's *infrastructure*. The traversal can be viewed as moving a single token around the cluster. The token carries the *id* of the cluster which created it and the maximum cluster-id observed so far (maximum-id). The cluster-id is used to distinguish between inter- and intra-cluster links, and the maximum-id is used to detect a cycle of clusters. The DFT token searches the cluster for a *new* link, a link never used by the algorithm. Doing so, the DFT token leaves behind a trail of active links, which leads from the cluster-head to the token.

Upon finding a *new* outgoing link, l, the token traverses over link l to a node, v, on the other end of l. If v belongs to the same cluster, the token returns to l's tail via the intree and the active path. Link l is then marked *killed*, and it will never again be traversed. If, on the other hand, the token arrives at a different cluster, the information it carries and the new selected link it has established enable the cycle detection to continue as described below.

3.4 Cycle Detection

For the sake of simplicity, we describe a rather inefficient algorithm for cycle detection. Since its complexity is, in general, not the bottleneck, we avoid discussing possible improvements (The improvements are a generalization of [Pete82, Dole82], with which our algorithm will perform optimally on rings).

After each cluster selects an outgoing link, the network contains at least one cycle which consists of two or more clusters (see figure 3.1). Let each cluster send its id on the cluster outgoing link. A cluster forwards another cluster-id only if it is larger than all the cluster-ids it has received in the past. Eventually, one and only one cluster in each cycle will receive the same cluster-id twice, thus detecting a cycle. The cluster-head that detected the cycle synchronizes the new cluster.

The operation of cycle detection is carried out by the cluster-heads. To implement it, each node receiving a message from a different cluster forwards it to its cluster-head through the cluster's intree. To forward a maximum-id from a cluster-head to the next cluster, the cluster-head broadcasts the maximum-id over the *infrastructure* of its cluster. All nodes retain a maximum-id variable, which is updated by the maximum-id of the broadcast message. The DFT token updates its maximum-id to the largest it encounters along its way. If a

cluster outgoing link has been selected, the blroadcast message is simply forwarded over the outgoing link to the next cluster.

When a cluster-head receives a maximum-id which is equal to its own, it has detected a cycle and the synchronization phase starts.

3.5 Cycle Contraction and Cluster Synchronization

In the synchronization phase the elected cluster-head notifies all the nodes in the clusters around the cycle of their new cluster-id. After receiving a positive acknowledgement that all the nodes have received the new cluster-id, the elected cluster-head starts the search for the cluster outgoing link. The notification is accomplished by broadcasting a message over the cluster *infrastructure*.

The purpose of the broadcast message is threefold; first, to mark all the selected outgoing links around the cycle, as *elementary* links; second, to combine the intrees of clusters around the cycle into one intree rooted at the elected cluster-head; and third, to notify all the nodes of the cluster of their new cluster-id. Upon receiving the first copy of the broadcast message, every node performs the following five operations: updates its own cluster-id; marks cluster-outgoing link (if it has one) as *elementary*; updates (if necessary) its TREE mark; forwards copies of the message over its *elementary* outgoing links; and acknowledges the reception of the message through the intree. Any duplicate copies of the message are discarded.

To merge all the trees around the cycle, we use the active paths in each cluster which lead from the cluster-heads to the head nodes of the selected outgoing links. The active paths are constructed during the DFT in the clusters outgoing link selection phase.

We notice that, if each node of a cluster that has an active outgoing link puts its TREE mark on the active link, then the incoming spanning tree is rerooted to the *head* node of the cluster outgoing link (see figure 3.2). (Note that the head node of the cluster outgoing link belongs to the next cluster on the cycle of clusters.) To obtain a directed intree which spans all the clusters around the cycle, we perform this operation in all the clusters around the cycle except for the elected cluster. Thus, the tree formed is rooted at the elected cluster-head. This intree is used by the nodes to notify their new cluster-head of the contraction termination.

To notify the cluster-head that all the nodes in the new cluster are aware of their new cluster-id, every node sends an acknowledgment as follows: After receiving the first copy and updating the TREE mark, every node sends an acknowledgment over all the *elementary* outgoing links aside from the TREE marked link. An acknowledgment is sent over the TREE marked outgoing link only after an acknowledgement has been received on all the *elementary*

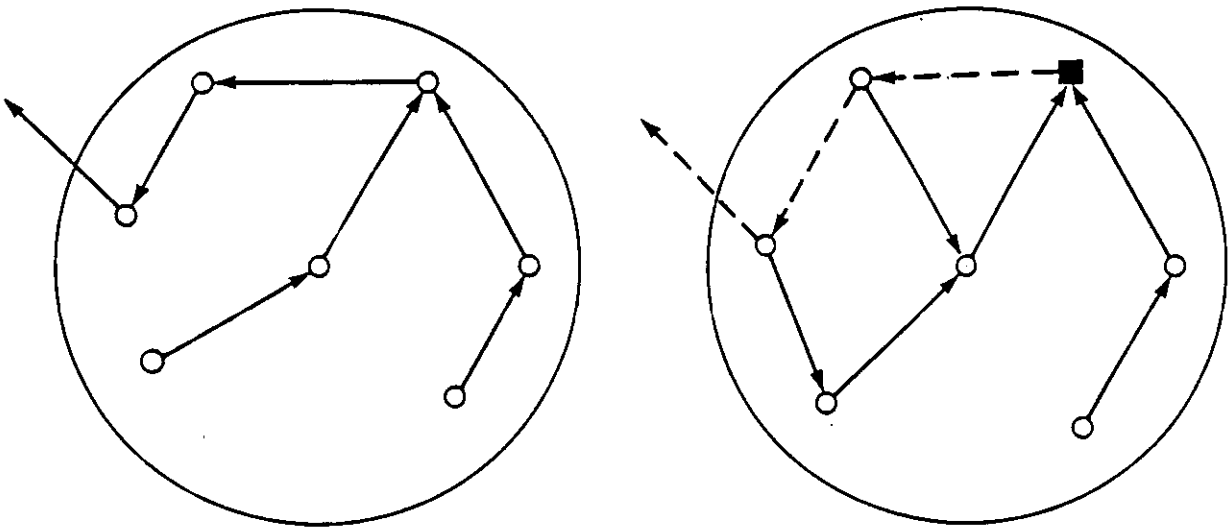

Figure 3.2: Rerooting an intree

incoming links.

When the elected cluster-head has received the acknowledgment message over all of its *elementary* incoming links, the contraction has terminated, and a new phase of cluster outgoing link selection is started.

The algorithm terminates when a cluster fails to select a cluster-outgoing link. This cluster spans the whole network, its cluster-head is the elected node and its maximum-id is the largest *id*.

4 Complexity of the Election Algorithm

Communication complexity analysis involves counting the total number of bits transmitted over all the network links. The communication cost of the algorithm has three components: the cost of cluster synchronizations, the cost of cycle detections and the cost of cluster-outgoing link selections.

We observe the following two facts.

Fact 1: The number of cycle detections is, at most, n-1.

Fact 2 : In a cluster of size k, there are, at most $2k$-1 *elementary* links.

Fact 1 holds because the contraction of a cycle strictly reduces the network size. Fact 2 follows from fact 1 and the observation that there exists a one-to-one correspondence between clusters and *elementary* links.

4.1 Cluster Synchronization Cost

lemma 1: The total cost of synchronizing the clusters is at most $O(n^2 \log n)$ bits.

proof: According to fact 1 there are, at most, $n-1$ cluster synchronizations. The synchronization messages propagate on the infrastructure of a cluster which contains, at most, $2n-1$ links. Since each message is of size $O(\log n)$ bits and there is a constant number of them in each phase, the result follows. |||

4.2 Cycle Detection Cost

lemma 2: The total cost of all cycle detections is at most $O(n^2 \log n)$ bits.

proof: A link forwards the same maximum-id, at most, four times for each cluster it belongs to. As a result, a link can forward, at most, $4n-4$ ids. Only links in the infrastructures forward ids, and since there are, at most, $2n-1$ of them, the result follows. |||

4.3 The Cost of Cluster Outgoing Link Selection

The cost involved in selecting outgoing links consists of the cost of killing intercluster links and the cost of the DFT's.

To kill link l, the algorithm transmits one message of $O(\log n)$ bits over l and a kill message of size O(1) bits over a path from the head of l to its tail. The kill message goes down the intree to the cluster-head and then along the active path to the tail of l. This node is distinguished from other nodes on the active path since it is the FOCAL POINT of the DFT. Thus, the killing of one link costs $O(n)$ bits. Since the algorithm kills, at most, m links, the killing of intercluster links adds up to, at most, $O(n \cdot m)$ bits.

As mentioned, the cost of one DFT on a network with n nodes and m links is $O(n \cdot m \log n)$ since there is one backtracking for each link of the network, and each backtracking costs $O(n \log n)$. The DFT operation is employed in the election algorithm to search the infrastructures of different clusters. Since there are at most twice as many links as nodes in an infrastructure, the cost incurred by each DFT of an infrastructure with n nodes is $O(n^2 \log n)$. As the DFT could be used $n-1$ times by the algorithm, the total cost of all DFTs

might be $O(n^3 \log n)$.

In order to avoid this increase of complexity we modify the algorithm as follows: A new phase is added, after a cycle is detected, in which the cluster-head of the largest size cluster around the cycle is elected to be the new cluster-head. The new cluster-head then proceeds with the cluster synchronization phase as described before. After synchronizing the cluster, the cluster-head resumes its DFT process from the previous stage, i.e., from the head node of its former cluster outgoing link, thus avoiding an unnecessary search of nodes that were already searched. The head node of its former cluster outgoing link is now the last node on the *active path.*

4.3.1 Cluster-Head Election

To elect a cluster-head of a largest size cluster (ties are resolved by cluster-ids), the cluster-head detecting the cycle sends an elect-message around the alternating sequence of active paths and intrees which form a directed cycle (see figure 3.1). On its way, the elect-message finds out which cluster has the greatest number of nodes and what the total number of nodes in all the clusters around the cycle is. This information is updated on the elect-message by the cluster-heads along the directed cycle.

Once the elect message returns to the originating cluster-head, the control of cluster synchronization is passed to the newly elected cluster-head. Any new and larger maximum-id which arrives at the cluster-head detecting the cycle during the election and the synchronization phase is held back by this node. It is forwarded to the new cluster-head upon the receiption of the synchronization broadcast.

To see that the above scheme does not add more than a constant factor to the complexity of a single DFT, we will quote the following lemma, introduced in a similar context by Gallager [Gall77]

lemma 3: For any given k , the number of clusters that own n/k nodes or more is, at most, k.

Corollary : The largest cluster to be captured by another cluster owns at most $n/2$ nodes, the next largest at most $n/3$, etc.

Thus, we arrive at the following:

lemma 4: The total cost of the DFT's is $O(n^2 \log n)$.

proof: The cost of traversing a cluster of size k is at most $k^2 \log n$ bits. Hence, the total cost is bounded by $\sum_{i=1}^{n} \left(\frac{n}{i}\right)^2$ messages. This series, in turn, is majorized by $\sum_{i=0}^{\lfloor \log n \rfloor} \left(\frac{n^2}{2^i}\right) \leq 2\ n^2$. Hence, the bit complexity of the depth first traversals is $O(n^2 \cdot \log n)$. |||

Adding the bit costs of all three components, we arrive at a total communication complexity of $O(n^2 \log n + n \cdot m)$ bits for the whole algorithm.

5 A Traversal Algorithm for Unidirectional Network of Finite Automata

Imagine the behavior of the election algorithm when it is started by a single node. In this case, the initiator spawns a process which visits all the nodes in the network and terminates. In the first subsection we show that the process can be viewed as a traversal process which explores the nodes one at a time. Furthermore, the traversal will be modified to terminate at the initiator. Hence, the traversal algorithm is also useful as a mechanism for efficient broadcast with feedback [Sega83] in a unidirectional network. It is this view of the election algorithm that leads us to derive a traversal algorithm that is to be described in this section.

In subsection 5.2 we show that the traversal can be implemented in a unidirectional network of finite automata while achieving the same communication complexity as the election algorithm (requiring a finite amount of memory per node and per link). The reader familiar with [Tarj72] will not fail to notice the similarity between the resulting traversal process and the biconnectivity algorithm.

5.1 A Traversal Algorithm for a Unidirectional Network with O(log n) Memory Bits per Node

Assuming that only one node initiates the election algorithm, we make the following observations:

First, at any given time, at most one cluster is searched, i.e., no messages are exchanged in any of the other clusters.

Second, all the clusters and their selected outgoing links form a simple directed path in which each cluster is a node and each link is a selected outgoing link. This path, called the *clusters active path*, occasionally closes on itself (see figure 5.1). When the path closes either a cycle of clusters is formed, or the last link on the path is an intracluster link (in the last cluster on the path, see figure 5.1, cases 2 and 1, respectively).

Third, consider the *clusters active path* when it does not close on itself. Then, the nodes at which the *clusters active path* enters clusters are the first nodes to be explored in each cluster. Therefore, if these nodes were selected as the cluster-heads of their clusters, the active paths of all the clusters would form a single contiguous path at all times.

We now modify the election algorithm according to the above observations namely, in each cluster the first node to be explored is elected as the cluster-head. Cycle detection (case 2 in figure 5.1) occurs when the token leaves one cluster C_1 (by traversing it's selected outgoing link for the first time) and arrives at another, previously explored, cluster C_2. Recalling the third observation, the cluster-head of C_2 is the "oldest" node on the newly formed cycle of clusters. This cluster-head (h) synchronizes the cycle of clusters and, is elected to be the cluster-head of the new cluster. Also, (from the third observation) the *active paths* around the cycle form a single contiguous path. If the synchronization phase is modified to leave all the active paths intact, the distinct DFT's may be considered as one DFT. Thus the newly elected cluster-head, h resumes a DFT which already has an *active path* going through all the clusters around the cycle. As a result, no node in the network is searched more than once during the whole algorithm. The cluster-head resumes the DFT at the node in which the cycle was detected (the LOOP node in figure 5.1, case 2).

The above variation of the election algorithm can be viewed as a traversal process. One node spawns the process which terminates at the initiator after visiting all the nodes in the network, one at a time. This traversal process can be further modified to work on a unidirectional network of finite automata, as we show in the next section.

5.2 A Finite Automata Unidirectional Traversal Algorithm

Two operations in the preceding traversal algorithm require $O(\log n)$ bits memory in each node. The first is distinguishing an intracluster from an intercluster link. The second (which occurs during the DFT of the infrastructure) is backtracking from the last node on the active path.

In the first operation, the $O(\log n)$ bits are used to distinguish between the case that a newly traversed link, l, is an intracluster link and the case that l closed a cycle of clusters. This operation was accomplished in the preceding algorithm by comparing the *id* on token with the cluster-id of the head node of l. To perform the operation without *ids*, we note that,

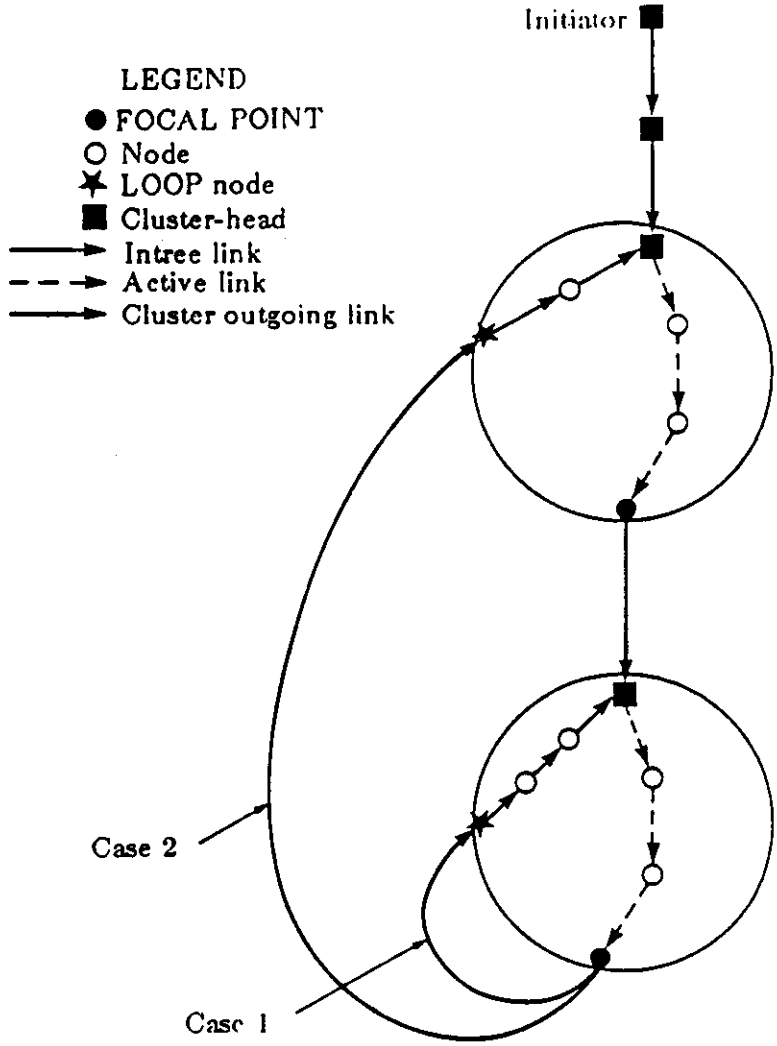

Figure 5.1: Clusters in the Traversal Algorithm

in both cases, l closed a directed cycle of nodes and links (composed of an active path followed by a path in an intree, see figure 5.1). In the first case, exactly one cluster-head resides on the cycle while, in the second case, at least two cluster-heads reside on the cycle (see figure 5.1).

We now explain how the token decides whether there is more than one cluster-head on the cycle. The last node on the active path in each cluster is marked FOCAL POINT (it is the FOCAL POINT of that cluster's DFT, see section 3.2). The head node of a newly traversed link l is marked LOOP if it is an already explored node. Upon arriving at a LOOP node, the token is sent around the cycle to find out whether there is more than one cluster-head on it. Since there is exactly one LOOP node on such a cycle, the walk around it utilizes a finite number of bits on the token. If exactly one cluster-head was found, the LOOP mark is removed. The token then walks around to the FOCAL POINT, kills l and resumes the DFT. If on the other hand, more than one cluster-head was found, a cycle of clusters was identified. The token then makes another trip around the cycle in order to synchronize its

clusters. On the second round, the token first erases all the FOCAL POINT marks. Second, it erases all the cluster-head marks, aside from the first. Third, the token modifies the intrees of all clusters, aside from the first one, to be rerooted at the LOOP node (in the same way done in the election algorithm). Notice that the *active path* of the first cluster lies between the first and the second cluster-head. Arriving at the LOOP node for the second time, the synchronization phase terminates. The LOOP mark is removed, the FOCAL POINT mark is put on, and the DFT is resumed.

Next we describe how the backtracking in the DFT can be performed without using node *ids*. To recognize the last node on the active path while backtracking, we use a solution to the following "last in the ring" problem: In a unidirectional ring of finite automata, design an algorithm by which a designated node, A, will distinguish the node preceding it, B, from all other nodes.

lemma 5: The upper and lower bounds on the bit complexity of the "last in the ring" problem are $O(n \log n)$.

Proof: A solution to the puzzle works in phases. Initially, all nodes except A are candidates for the position. In each phase we eliminate half of the remaining candidates by sending the token around the ring, alternately marking the candidate nodes even or odd. In each phase the token remembers the parity of the candidate preceding A. Thus, the token can eliminate all candidates (in the next phase) whose parity differs from the parity of the desired node. The token will detect the last phase when only one node is marked.

If all n links have seen fewer than $\log n$ bits, then at least two of them have seen the same sequence, which implies that the node preceding the designated one has to be at an equal distance from both. This is a contradiction; hence, $O(n \log n)$ is also the lower bound. ill

The communication cost of the cycle detection and synchronization phases is not increased in the modified algorithm. The cost of the cluster outgoing link selection phases is $O(n^2 \log n)$ bits since the DFT searching for outgoing links requires one backtracking for each link of the *infrastructure*. Thus, the communication complexity of the traversal algorithm is the same as that of the election algorithm.

The following lemma suggests that this algorithm may be optimal.

lemma 6 : $\Omega(n \cdot m)$ bits is the lower bound to a single token traversal of a unidirectional graph.

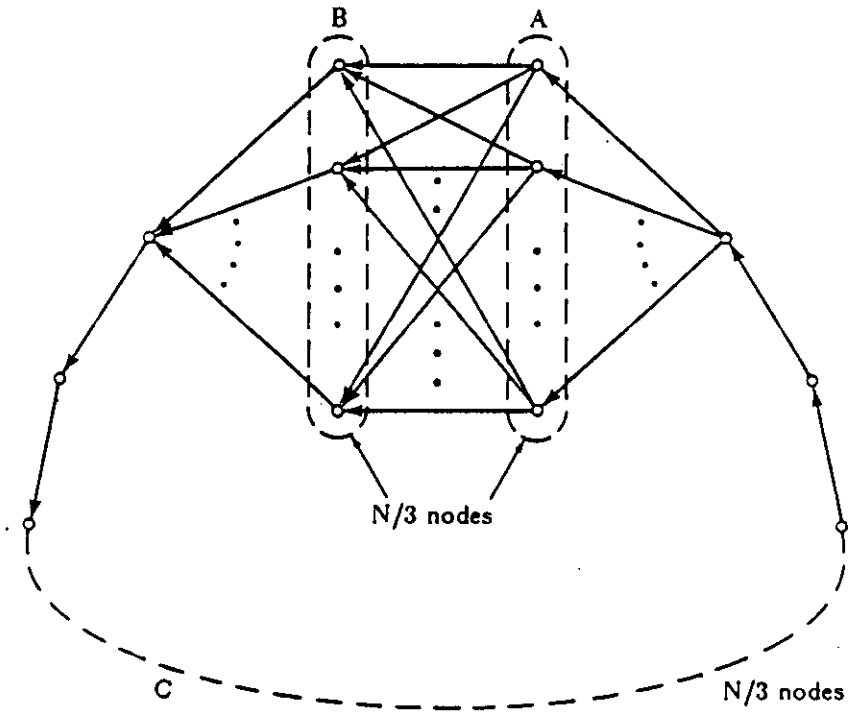

Figure 5.2: A network for the $\Omega(n \cdot m)$ lower bound

Proof : (by example, figure 5.2). The result follows since each traversal of a link from A to B must be followed by a traversal of the path C. |||

6 Concluding Remarks

The intree and outtree produced by the election algorithm can be used in a mechanism to simulate any bidirectional distributed algorithm on a strongly connected unidirectional network. If a problem has a bit complexity $O(P(n))$ on a bidirectional network, then its complexity on the unidirectional network is upper bounded by $O(n \cdot P(n) + n^2 \cdot \log n + n \cdot m)$.

As shown in [Gafn84a], any algorithm which requires common knowledge is equivalent to an election algorithm. Therefore, we expect the election and traversal algorithms to serve as building blocks in many unidirectional network algorithms. An example of such an application, which involves termination detection, can be found in [Misr83].

An interesting observation is that the amount of communication in the unidirectional election algorithm, $O(n \cdot m + n^2 \cdot \log n)$ bits, is n times the number of messages in the optimal bidirectional algorithm. We would obtain the same cost if we were to simulate the bidirectional algorithm, [Gall83] with each acknowledgement charged as n bits, on a unidirectional network. Together with lemma 6, this leads to the conjecture that our election algorithm is as efficient as possible in terms of communication cost.

Acknowledgements

We are in debt to Prof. R. Gallager for bringing [Gall77] to our attention.

References

[Burn80] J. E. Burns, "A Formal Model for Message Passing systems," 91, Indiana Univ., Bloomington (May 1980).

[Dole82] Danny Dolev, Maria Klawe, and Michael Rodeh, "An O(n log n) Unidirectional Algorithm for Extrema Finding in a Circle," *Journal of Algorithm* **3**, pp.245-260 (1982).

[Fred83] Greg N. Frederickson and Nancy A. Lynch, "The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring," (Nov. 83). Detailed Abstract.

[Gafn84a] Eli Gafni, Mike Loui, Parsoon Tawiri, Doug West, and Shmuel Zaks, "Lower Bounds on Common Knowledge in Distributed Algorithms, with Applications," Preprint, Univ. of Illinois (January 1984).

[Gafn84b] Eli Gafni and Willard Korfhage, "Election on A Unidirectional Eulerian Graph," , UCLA Computer Science Department (March 1984). Preprint.

[Gall76] Robert G. Gallager, "A Shortest Path Routing Algorithm with Automatic Resynch," , M.I.T. (March 1976). Unpublished note.

[Gall77] Robert G. Gallager, "Finding a Leader in a Network with O(E) + O(N log N) Messages," , M.I.T. (1977). Unpublished Note.

[Gall83] Robert G. Gallager, Pierre A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Trans. Program. Lang. Syst.* **5**, pp.66-77 (Jan 1983).

[Humb83] Pierre A. Humblet, "A Distributed Algorithm for Minimum Weight Directed Spanning Trees," *IEEE Trans. Comm.* **COM-31**(6), pp.756-762 (June 1983).

[Koze79] D. Kozen, "Automata and Planar Graphs," IBM Research Report RC 7604, IBM (Aprl 1979).

[Lync81] N.A. Lynch and M.J. Fischer, "On describing the behavior and implementation of distributed systems," *Theoretical Computer Science* **13**, pp.17-43 (1981).

[Misr83] J. Misra, "Detecting Termination of Distributed Computations Using Markers," pp. 290-295 in *Proceedings of the ACM Symp. on Principles of Distributed Computing*, Montreal (August 1983).

[Pach82] J. Pachl, E. Korach, and D. Rotem, "A Technique for Proving Lower Bounds for Distributed Maximum-Finding algorithms," pp. 378-382 in *Proceedings of the 14th Ann. ACM Symp. on Theory of Computing*, San Francisco CA (1982).

[Pete82] Gary L. Peterson, "An O(n log n) Unidirectional Algorithm for the Circular Extrema Problem," *ACM Trans. Program. Lang. Syst.* **4**(4), pp.758-762 (October 1982).

[Sega83] Adrian Segall, "Distributed Network Protocols," *IEEE Transactions on Information Theory* **IT-29**(1) (January 1983).

[Shan71] H. S. Shank, "Graph Properties Recognition Machines," *Mathematical Systems Theory* **5**, pp.45-49 (1971).

[Tarj72] Robert Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM J. Comput.* **1**, pp.146-160 (1972).

Originally published in:

ACM Symposium Proceedings, Principles of Distributed Computing

Vancouver, British Columbia

August 1984