FROM RINGS TO COMPLETE GRAPHS
Θ(n log n) TO Θ(n) DISTRIBUTED LEADER ELECTION

by

H. Attiya¹, N. Santoro² and S. Zaks

Technical Report #446

February 1987

¹ Dept. of Computer Science, Hebrew University, Jerusalem, Israel
² School of Computer Science, Carleton University, Ottawa, Canada
³ Dept. of Computer Science, Technion, Haifa, Israel; the work of this author was supported part by a Technion Research grant No. 120-0641.
FROM RINGS TO COMPLETE GRAPHS -
\( \Theta(n \log n) \) TO \( \Theta(n) \) DISTRIBUTED LEADER ELECTION

Hagit Attiya\(^1\), Nicola Santoro\(^2\) and Shmuel Zaks\(^3\)

ABSTRACT

We study the message complexity of the problem of distributively electing a leader in chordal rings. Such networks consist of a basic ring with additional links; the extreme cases being the oriented ring and the complete graph with a full sense of direction. We present a general election algorithm for these networks, and prove its optimality. As a corollary, we show that \( O(\log n) \) chords at each processor suffice to obtain an \( O(n) \) algorithm; this improves and extends a previous work, where \( O(n) \) algorithm was suggested for the case where all \( n-1 \) chords exist (the oriented complete network).

1. INTRODUCTION

1.1. General.

The problem of electing a leader has been widely studied in the literature of distributed networks. In this environment, the processors in the network have to identify one of them as a leader, in order to resolve a certain conflict of which some (possibly all) of them are aware. This problem arises in all the situations where a single processor is needed to control a certain function. A better understanding of these problems is crucial in the study of these networks.

This problem exhibits many of the crucial aspects of distributed computing. The factors determining the complexity of election algorithms include the structure of the underlying network, as well as the amount of information available at each processor; this information may be local (regarding the processor’s neighbors) or global (regarding the entire graph).

\(^1\) Department of Computer Science, Hebrew University, Jerusalem, Israel.
\(^2\) School of Computer Science, Carleton University, Ottawa, Canada.
\(^3\) Department of Computer Science, Technion, Haifa, Israel; the work of this author was supported in part by a Technion Research grant no. 120-0641.
1.2. The Model

Our model consists of \( n \) processors \( p_0, p_1, \ldots, p_{n-1} \) arranged on a ring such that \( p_i \) is connected to \( p_{i+1 \bmod n} \) and to \( p_{i-1 \bmod n} \). By adding chords to the ring we get chordal rings. The processors can communicate only by sending messages asynchronously along ring edges or chords. Every processor \( p_i \) has an identity \( id_i \) (chosen from some infinite totally ordered set \( ID \)) which is known only to itself. It is assumed that every message will reach its destination after a finite but unbounded delay, that each communication line preserves the order of the messages sent on it, and that all the identities are distinct.

An algorithm is a program that contains three types of executable statements: local computations, message send and message receive statements. An execution is a sequence of events, each being one of the above three statements. The aim is to design an algorithm such that when executed at every processor, exactly one processor will be selected as the \textit{leader}. The algorithm may be independently started by any subset of the processors. The complexity measure for an algorithm is the maximum number of messages sent during any possible execution. This measure is natural for rings, assuming negligible internal processing time (see, e.g., [DKR], [KMZ], [P]).

1.3. Sense of direction

An important parameter determining the message complexity is \textit{sense of direction} (see [S]). Sense of direction refers to the capability of a processor to distinguish between its adjacent communication lines, according to some globally consistent scheme. In a ring network this property is usually referred to as \textit{orientation}, which expresses the processor's ability to distinguish between left and right, where 'left' means the same to all processors. Sense of direction in a complete network was defined in [S] as the knowledge of some predefined Hamiltonian cycle, and the existence of a label on each link at each processor, that represents the distance along this cycle to the processor at the other end of the link. This oriented complete network can also be viewed as a directed ring on which all possible chords were added. An example for the case \( n=6 \) is shown in the figure.
These two extreme cases of the ring and the complete network demonstrate the impact of this notion of sense of direction on the communication complexity. Electing a leader in oriented rings requires at most $O(n \log n)$ messages ([DKR],[F],[P]), matching the $\Omega(n \log n)$ lower bound ([B],[PKR]). In a complete network some problems have a high message complexity exactly because of the fact that a processor does not have a sense of direction, and many edges have therefore to be used in order to get to a desired destination. It was shown in [KMZ] that in an unoriented complete network - namely, when no local information is known, and all the edges adjacent to a given processor are indistinguishable - the message complexity of electing a leader is $\Omega(n \log n)$ in the worst case. In [LMW] (see also [SSU]) the oriented complete network is studied, and it is shown how to elect a leader using at most $O(n)$ messages. This result can be extended to the case where, besides the above, a processor cannot distinguish between its left and right (see [SUZ]).
These results show the impact of knowledge about the organization of the network on the communication cost of elections. The small diameter of the complete network and the large diameter of the ring are the main reason for the difference between the $\Omega(n \log n)$ lower bound for oriented rings and the $O(n)$ algorithm for the oriented complete networks.

1.4. Results

Define the chordal ring (see [AL]) $C_n <i_1, \cdots, i_s> = <V, E>$, where $i_1, i_2, \cdots, i_s \in \{2, \cdots, n-2\}$, as the undirected graph with set of vertices $V = \{0, 1, \cdots, n-1\}$ and set of edges $E = E_1 \cup E_2$, where $E_1 = \{(i, i+1) | i=0, \cdots, n-1\}$ and $E_2 = \{(i, i+j) | j = i_1, \cdots, i_s, i = 0, \cdots, n-1\}$ (all arithmetic is modulo $n$). The edges in $E_1$ are called ring edges and the edges in $E_2$ are called chords. For a chord $(i, i+j)$ call $j$ the chord distance. Note that node numbering is external and it is unknown to the processors (otherwise, for example, processor $n$ could be the elected leader). However, processor $i$ knows for each edge whether it is a ring edge or a chord (with the appropriate chord distance). In particular, $C_n <2, \cdots, n-2>$ is the complete oriented network considered in [LMW], and $C_n <$ is the oriented ring. The results in [LMW] and [DKR,PK] determine the complexity of elections in these two extremal cases. In this work we explore the complexity of elections in the intermediate range. Note that in chordal rings without sense of direction an $O(n \log n)$ election algorithm can be derived from the algorithm in [KMZ] or similar ones.

A simple technique to reduce the cost of election algorithms on chordal rings is developed and its complexity is analyzed. Using this technique we devise an algorithm that elects a leader in the chordal ring $C_n <2, \cdots, s>$ using $O(n + \frac{n}{\alpha} \log \frac{n}{\alpha})$ messages. As a corollary we show that $O(\log n)$ additional chords suffice in order to conduct elections using $O(n)$ messages, thus improving the result in [LMW] (we note that an $O(n)$ algorithm can be shown also for the chordal ring $C_n <2, 4, 8, \cdots, 2^{[\log n]}>$; the proof is left to the reader). For the chordal ring $C_n <\alpha, 2\alpha, \cdots, s\alpha>$ we present an algorithm that uses at most $O(n + \alpha^2 + \frac{n}{\alpha} \log \frac{n}{\alpha^2})$ messages. We discuss the optimality of the results for various cases.
2. THE ALGORITHM

We first describe and analyze the algorithm in the case where we are given a chordal ring, in which sending a message to distance $d$ requires at most $\left\lfloor \frac{d}{c} \right\rfloor$ hops, for some constant $c$. We delay the concrete description of how this is done to a later stage.

The basic idea of the algorithm is very simple and resembles the algorithm given in [LMW]. We run an election algorithm on the ring and use the chords to efficiently communicate information between processors. We show the algorithm for oriented chordal rings. However, the algorithm can be modified for unoriented chordal rings along the lines of [SUZ].

The algorithm is a modification of the election algorithm for bidirectional rings ([F]). The algorithm proceeds in phases. At each phase, some of the processors are candidates for election, and are active, i.e. generate messages. For simplicity of presentation we assume all processors are initially active; the modification of the algorithm and the analysis for the general case is easy. A processor is a local maximum if its identity is larger than these of both its neighboring active processors. Only the local maxima of the current phase remain active in the next phase. A processor that is not a local maximum becomes passive and will send exactly one additional message. At each phase every active processor exchanges its identity with its two neighboring active processors. Each processor holds the distances to the next active processors to its left and right. If a processor is a local maximum it generates a message to update the next active processor to its left of its distance from it. This message is forwarded by all currently active processors that are not local maxima. As the algorithm proceeds, the number of the candidates, i.e. active processors, is reduced, until only one is left which becomes the elected leader.

Each processor holds its own identity in the variable $my_id$, and its neighbors' identities in $left_id$ and $right_id$. The distance to the next active processor to its left (right) is kept in $next_left$ ($next_right$). It also keeps a boolean variable $local\_maximum$. We use two commands, SEND and RECEIVE to handle communication, with the following semantics:

SEND(m, d) - Send the message m to the processor at distance d. This might require up to $\left\lfloor \frac{d}{c} \right\rfloor$ hops;
when $d$ is positive (negative) the message is sent to the right (left). In general, SEND is not a primitive procedure, while in [LMW] each SEND operation requires a single message transfer.

RECEIVE($m$) - Wait for (and receive) a message $m$, and assign values to variables according to $m$.

A message $m$ always contains an identity and an integer indicating distance along the ring; namely, $m \in ID \times \{1, 2, \ldots, n-1\}$.

Each processor executes the following algorithm.

```
begin
  next_right := 1;
  local_maximum := true;
  SEND((my_id, 1)); { send to your right }
  RECEIVE((left_id, next_left)); { receive from your left }
  repeat
    SEND((my_id, next_left), -next_left); { send to your left }
    RECEIVE((right_id, next_right)); { receive from your right }
    if (my_id $\geq$ left_id) and (my_id $\geq$ right_id) then
      { local maximum, remain active }
      begin { initiate message to measure distance to next active processor }
        SEND((my_id, next_right), next_right);
        RECEIVE((left_id, next_left))
      end
    else begin
      local_maximum := false;
      RECEIVE((left_id, next_left));
      SEND((left_id, next_left+next_right), next_right)
    end
  until (not local_maximum) or (left_id=my_id);

  if left_id=my_id then "elected"
end.
```

3. CORRECTNESS AND COMPLEXITY ANALYSIS

As mentioned before, our algorithm is based on the one in [F], only that we use the chords between active processors instead of communicating through the ring edges. We therefore omit the formal proof of correctness. We discuss now the message complexity of the algorithm.
The execution of the algorithm can easily be shown to proceed in phases. All processors start at phase 1. A processor at phase $p$ moves to phase $p+1$ if it passes the local maximum test (in the repeat loop), otherwise it stops being active. An active processor $P_i$ sends exactly two messages at each phase - one towards each of its left and right neighboring active processors. Otherwise, it sends exactly one more message (in the else part of the if statement) that is used by its right active neighbor to measure the distance to its left active neighbor, which enables it to bypass $P_i$ in subsequent phases. To obtain an upper bound for the number of message hops used by the algorithm we have to bound the number of phases and the amount of work done by the active processors at each phase.

Let $n_p$ denote the number of active processors at the end of phase $p$ ($n_0 = n$). Since at most one out of every two consecutive active processors remains active in the next phase, we have

$$n_{p+1} \leq \frac{n_p}{2}$$

for $p = 0,1,\ldots$, and therefore

$$n_p \leq \frac{n}{2^p}.$$  

This implies that the number of phases is at most $\left\lceil \log_2 n \right\rceil$. Let $d_1, \ldots, d_n$ be the distances between the consecutive active processors in phase $p$. Clearly $\sum d_i = n$. Since at each phase an active processor sends exactly two messages, the total number $M_p$ of messages sent (and received) during phase $p$ satisfies

$$M_p = \sum_{i=1}^{n} 2 \left[ \frac{d_i}{c} \right] \leq 2 \sum_{i=1}^{n} \left[ \frac{d_i}{c} \right] + 1 = 2 \left[ \frac{n}{c} + n_p \right] \leq 2 \left[ \frac{n}{c} + \frac{n}{2^p} \right].$$

A processor that is not a local maximum sends exactly one more message, thus the total number of these messages is bounded by $n$. Therefore, the total number $M$ of messages sent by the algorithm satisfies

$$M \leq n + \sum_{p=1}^{\left\lceil \log_2 n \right\rceil} M_p \leq n + \sum_{p=1}^{\left\lceil \log_2 n \right\rceil} 2 \left[ \frac{n}{c} + \frac{n}{2^p} \right] = n + 2 \frac{n}{c} \left\lceil \log_2 n \right\rceil + 2n \sum_{p=1}^{\left\lceil \log_2 n \right\rceil} \frac{1}{2^p} \leq 2 \frac{n}{c} \log_2 n + 3n. \quad (*)$$
4. EXAMPLES

As a first example we consider the simplest case, $C_n < 2, \ldots, t >$, where $t$ consecutive chords were added. To send a message in $C_n < 2, \ldots, t >$ using $\left\lceil \frac{d}{t} \right\rceil$ hops, we first send the message to distance $\left\lceil \frac{d}{t} \right\rceil$ using $\left\lfloor \frac{d}{t} \right\rfloor$ chords of distance $t$, and if necessary (when $d \mod t \neq 0$) an additional message on a chord to distance $d \mod t$. By the previous discussion the total cost of elections in $C_n < 2, \ldots, t >$ is $2^n \log_2 n + 3n$ messages in the worst case.

When $t > 3.3 \log_2 n$ this amounts to an algorithm using less than $3.62n$ messages, thus improving [LMW] (where the bound is derived using all possible chords, namely $t=n-1$). On the other hand, for $1 \leq t \leq \alpha(\log n)$, we get a continuous range of complexities $O(n \log n)$ to $O(n)$.

The analysis can be further refined in this case. To see this, note that in the first $\log t$ rounds the distance a message has to travel is bounded by $t$, and this can be made in one step. Hence for $p \leq \log t$ we have

$M_p \leq 2n_p \leq \frac{n}{2^p - 1}.
$

Substituting this in the above estimation for $M$ results in an upper bound of $2^n \log_2 n + 3n$ messages.

As a second example consider the more general chordal ring $C_n < \alpha, 2\alpha, \ldots, t \alpha >$, where $\alpha \cdot t$ divides $n$. This ring can be decomposed into $\alpha$ rings, each being a $C_n < 2, \ldots, t >$ ring. As a first phase, run the algorithm on each of these rings. By the previous analysis, the cost for each of these rings is

$O\left( \frac{n}{\alpha} + \frac{n}{\alpha^2} \log \frac{n}{\alpha^2} \right).
$

Since we run the algorithm in parallel on $\alpha$ such rings, the total cost so far is

$O(n + \frac{n}{t} \log \frac{n}{\alpha^2}).
$

We now have $\alpha$ locally elected candidates, among which we conduct election to find a leader. A message from one processor to another at distance $d$ costs at most $\left\lfloor \frac{d}{\alpha} \right\rfloor + \alpha$ messages. We can show, in a way similar to the previous example, that the total cost $M_p$ of a phase $p$ satisfies
and hence the total cost of this stage satisfies

\[ M = \sum_{\alpha=1}^{\log n} M_\alpha = O\left(\frac{n}{\alpha} \log \alpha + \alpha^2\right). \]

The total cost of the algorithm for the ring \( C_n \langle \alpha, 2\alpha, \ldots, t \alpha \rangle \) is therefore

\[ O(n + \frac{n}{\alpha} \log \frac{n}{\alpha} + \frac{n}{\alpha} \log \alpha + \alpha^2) = O(n + \alpha^2 + \frac{n}{\alpha} \log \frac{n}{\alpha}). \]

5. LOWER BOUNDS AND DISCUSSION

In this paper, we explored the cost of elections in a distributed chordal ring system. We presented an algorithm that uses at most \( O(n + \frac{n}{\alpha} \log \frac{n}{\alpha}) \) messages for the chordal ring \( C_n \langle 2, \ldots, t \rangle \), which yields a whole spectrum of topologies and relative complexities. A closer look at the algorithm reveals that if \( k \) processors initiate the algorithm, at most \( O(n + \frac{n}{\alpha} \log \frac{k}{\alpha}) \) messages are sent.

The following simple argument shows that the algorithm is optimal for the case \( \alpha=1 \). Conversely, assume we are given an algorithm that elects a leader in \( C_n \langle 2, \ldots, t \rangle \) with \( o\left(\frac{n}{\alpha} \log \frac{n}{\alpha}\right) \) messages. We can reduce it to an algorithm that elects a leader on a ring of size \( \frac{n}{\alpha} \) with \( o\left(\frac{n}{\alpha} \log \frac{n}{\alpha}\right) \) messages, thus contradicting the known \( \Omega\left(\frac{n}{\alpha} \log \frac{n}{\alpha}\right) \) lower bound for rings of size \( l \) ([B, PKR]). The reduction can be done in the following way. Consider the ring \( C_n \langle 2, \ldots, t \rangle \) with processors \( p_0, \ldots, p_{n-1} \). Define a new ring \( C_n \langle q_1, \ldots, q_n \rangle \) numbered \( q_1, q_2, \ldots, q_n \). To elect a leader in \( C_n \langle q_1, \ldots, q_n \rangle \), take the algorithm given for \( C_n \langle 2, \ldots, t \rangle \) and run it, letting processor \( q_i \) imitate processors \( p_{(i-1)n}, \ldots, p_{in-1} \), for \( i = 1, 2, \ldots, \frac{n}{t} \) (with adequate assignment of identities).
For the more complicated chordal ring $C_n <\alpha, 2\alpha, \cdots, t\alpha>$, we presented an election algorithm that uses at most $O(n + \alpha^2 + \frac{n}{\alpha} \log \frac{n}{\alpha})$ messages in the worst case. For constant values of $\alpha$ the optimality of this algorithm follows like in the previous case. For a general $\alpha$ we can show that $\Omega(n \log n)$ messages are necessary. We use a simple reduction: assume we are given an election algorithm for $C_n <\alpha, 2\alpha, \cdots, t\alpha>$ which uses $o(n \log n)$ messages. Then separate the ring $C_n <\alpha, 2\alpha, \cdots, t\alpha>$ into $\frac{n}{\alpha}$ rings of size $\alpha$ each. If the algorithm uses $o(n \log n)$ then there is a ring of size $\alpha$ where the algorithm sends $o(\alpha \log n)$ messages. Since we can simulate the chordal ring algorithm on a ring of size $\alpha$, this contradicts the known lower bound of $\Omega(l \log l)$ for rings of size $l$ ([IB, PKR]).

The results presented in this paper give a better understanding of the impact of the network topology on the efficiency of election algorithms. In networks where efficient election algorithms exist a processor knows exactly with which node it should compete in the next phase and it can cheaply make this competition. In contrast, in the complete network a processor does not know which edge leads to the node with which it should compete next (this is the essence of the lower bound proof in [KMZ]), whereas in the ring it knows exactly with which node it has to compete but communicating with this processor is expensive.

Acknowledgement: We would like to thank Ziv Harpaz for helpful comments.

REFERENCES


