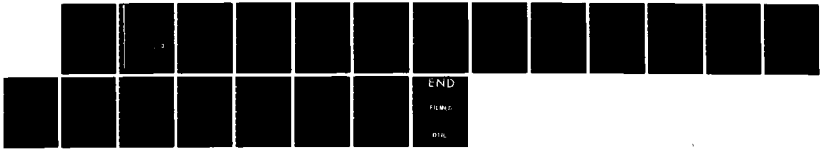
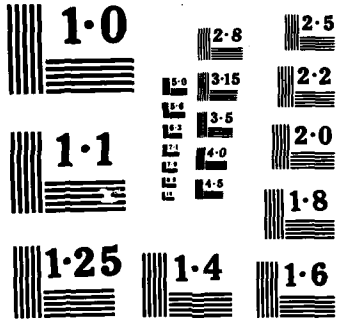
AD-A156 241 A GENERAL LOWER BOUND FOR ELECTING A LEADER IN A RING (U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE G N FREDERICKSON ET AL. MAR 85 1/1

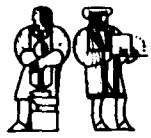
UNCLASSIFIED MIT/LCS/TM-277 N00014-83-K-0125 F/G 12/1 NL





(2)

LABORATORY FOR COMPUTER SCIENCE



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-277

A GENERAL LOWER BOUND FOR ELECTING A LEADER IN A RING

Nancy A. Lynch

Greg N. Frederickson

March 1985

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MIT/LCS/TM-277	2. GOVT ACCESSION NO. AD-A156241	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A General Lower Bound For Electing A Leader In A Ring		5. TYPE OF REPORT & PERIOD COVERED Interim research ~~May~~ 1985 6. PERFORMING ORG. REPORT NUMBER MIT/LCS/TM-277
7. AUTHOR(s) Greg N. Frederickson and Nancy A. Lynch		8. CONTRACT OR GRANT NUMBER(s) DARPA/DOD N00014-83-K-0125
9. PERFORMING ORGANIZATION NAME AND ADDRESS MIT Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA/DOD 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE March 1985 13. NUMBER OF PAGES 16
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* ONR/Department of the Navy Information Systems Program Arlington, VA 22217		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release, distribution is unlimited.		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)* Unlimited		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)* Leader election, distributed algorithms, lower bounds, synchronous algorithms, message complexity and symmetry		

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A lower bound of $\Omega(n \log n)$ messages is proved, for the problem of electing a leader in a ring of n processors. Unlike in previous work, the value of n is arbitrary, and not constrained to be a power of 2. The result is proved for comparison algorithms, but previously known techniques using Ramsey's theorem show that the result applies to other types of algorithms as well.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

A GENERAL LOWER BOUND FOR ELECTING A LEADER IN A RING

Greg N. Frederickson

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

and

Nancy A. Lynch

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

March, 1985

ABSTRACT

A lower bound of $\Omega(n \log n)$ messages is proved, for the problem of electing a leader in a ring of n processors. Unlike in previous work, the value of n is arbitrary, and not constrained to be a power of 2. The result is proved for comparison algorithms, but previously known techniques using Ramsey's theorem show that the result applies to other types of algorithms as well.

Keywords: Leader election, distributed algorithms, lower bounds, synchronous algorithms, message complexity and symmetry.

The work of the first author was supported by the National Science Foundation under grant DCR-8320124. The work of the second author was supported by NSF Grant No. DCR-8302391, U.S. Army Research Office Contract #DAAG29-84-K-0058, Office of Naval Research Contract #N00014-85-K-0168, and Advanced Research Projects Agency of the Department of Defense Contract #N00014-83-K-0125.

1. Introduction

We prove a lower bound of $\Omega(n \log n)$ messages, for the problem of electing a leader in a ring of n processors. The processors are assumed to be identical except that each has its own unique identifier. Each processor is allowed to know n, the size of the ring. Processors can communicate only with their neighbors in the ring, by sending messages. We assume that execution of the system is synchronous, and that all processors begin their algorithms at the same time. Unlike in previous work, the value of n is arbitrary, and not constrained to be a power of two. We prove the result for comparison algorithms, but previously known techniques using Ramsey's theorem show that the result holds under different assumptions.

The problem of determining the number of messages required to elect a leader in a ring has attracted considerable attention [B,CR,DKR,FL,GHS,HS,IR,L,P,V]. There are deterministic algorithms for the synchronous model [FL,V] which use O(n) messages, but which require large amounts of time and which use the processor ID's to count. The best deterministic algorithms which use comparisons of ID's only use O(n log n) messages [HS,GHS,B,DKR,P]. These algorithms work for both the synchronous and asynchronous models. On the other hand, a lower bound of $\Omega(n \log n)$ (with constant 1/4) has been proved for the number of messages required if communication is asynchronous [B]. If communication is synchronous, and either (a) the algorithm is restricted to using comparisons on ID's only, or (b) the algorithm is time-bounded independently of the size of the set from which the ID's are chosen, then an $\Omega(n \log n)$ lower bound (with constant 1/2) has been established [FL]. However, the results of [FL] require the severe limitation that the value of n is a power of 2. In this paper, we show how to prove an $\Omega(n \log n)$ lower bound for the same models, but for arbitrary values of n.

Complexity results are often first proved for powers of 2, and then extended to arbitrary values in a straightforward way. The lower bound proofs of [FL] do not appear to extend in such a simple manner. Those lower bounds are proved for a particular highly symmetric circular configuration of ID's, where the symmetry is based on having a number of processor ID's that is precisely a power of 2. One might construct a symmetric configuration for a power of 2, and then try to extend it is some way for the extra processors. However, this strategy introduces special treatment for the extra processors. This special treatment might change the behavior of the algorithm entirely, perhaps allowing some processor to become elected easily.

We introduce several nice techniques beyond those in [FL]. Two of these techniques are of particular interest. First, we generalize the notion of "chains" from [FL], as a way of describing limitations on information flow during a computation. Second, we develop a hierarchical assignment of processor ID's that

creates considerable "replication symmetry" around the ring. Such an assignment can be produced for arbitrary values of n. The fact that there are rings of every size having this type of symmetry seems to us to be quite interesting.

Our new lower bound still requires that either assumption (a) or (b) above hold. We give the proof for case (a) only, with the result for case (b) following as in [FL], using a reduction defined via Ramsey's theorem.

The constant we obtain for our general $\Omega(n \log n)$ bound is $1/(18 \log_2(9))$, which is much smaller than the constant of 1/2 obtained in [FL]. Thus, for the special case where n is a power of 2, the results of [FL] still give the strongest known lower bounds.

The remainder of the paper is organized as follows. Section 2 contains the formal model and problem statement. Section 3 contains the general theory used to prove the lower bound for comparison algorithms. Namely, it is shown that, in a ring with certain "replication symmetry" properties, many messages are required in order to distinguish certain processors. Section 4 contains a construction of a ring with such replication symmetry properties. Section 5 integrates the results of Sections 3 and 4, thus yielding the lower bound result for comparison algorithms. As a corollary, we obtain a corresponding lower bound for time-bounded algorithms. Section 6 describes remaining questions.

2. Formal Model and Problem Statement

In this section, we describe the formal model we use for our lower bounds. The contents of this section are summarized from [FL], and the interested reader is referred to the other paper for additional details.

2.1. Algorithms

All processors are assumed to be identical except that each has its own unique identifier, chosen from an *ID space* X, a totally ordered set. We assume that all processors begin their election algorithms at the same time. Each processor is modelled as an automaton that behaves as follows. Initially, the processor has an ID from an ID space X recorded in its state. At each round, the processor examines its state and decides whether to send a message to each of its neighbors, and what message to send. Then each processor receives any messages sent to it in that round. The processor uses its current state and these new messages to update its state. Certain of the states are designated as "elected" states.

It may be assumed, without loss of generality, that algorithms are in a certain normal form. In this normal form, the state of each processor records exactly its initial ID and the history of messages received, and each

message which is sent contains the entire state of the sending processor. We represent such history information by means of LISP S-expressions. The S-expressions that arise during computation are of a special type, which we will call the *well-formed* S-expressions over X. The atoms are $x \in X$ and NIL. The *well-formed* S-expressions over X are just the following: (1) the elements of X, and (2) those of the form (s_1,s_2,s_3), where s_2 is a well-formed S-expression over X, and s_1 and s_3 are either well-formed S-expressions over X or NIL. Let $\mathcal{S}(X)$ denote the set of well-formed S-expressions over X.

The initial states will just be the ID's $x \in X$. Each message will contain exactly the state of the sending processor. When a processor in state s receives messages s_1 and s_2 from its counterclockwise and clockwise neighbors respectively, its new state will be the S-expression (s_1,s,s_2). (If no message is received from a neighbor, the atom NIL is used in place of s_1 or s_2 as a placeholder.) To complete the algorithm specification, we define a function which determines when messages are to be sent in either direction, and a designation of which states indicate that the processor has been elected. Thus, an *algorithm* over X is a pair (E,μ), where $E \subseteq \mathcal{S}(X)$ is the set of *elected states*, and μ, a mapping $\mathcal{S}(X) \times \{\text{counterclockwise,clockwise}\} \rightarrow \{\text{yes,no}\}$, is the *message generation function*. We assume that the set E of elected states is "closed": if $s \in E$ and $s_1, s_2 \in \mathcal{S}(X) \cup \{\text{NIL}\}$, then $(s_1,s,s_2) \in E$. Thus, once a processor has been elected, it will remain elected.

2.2. Executions

To facilitate discussion, we index the processors in the ring clockwise, as 0,...,n-1. We count indices modulo n. A *ring* of size n over ID space X is an n-tuple of elements of X, giving the initial ID's of the processors 0,...,n-1, in order. A *configuration* of size n is an n-tuple of S-expressions in $\mathcal{S}(X)$, representing the states for the n processors. A *message vector* of size n is an n-tuple of ordered pairs of elements of $\mathcal{S}(X) \cup \{\text{null}\}$. It represents the messages sent counterclockwise and clockwise by each of the n processors.

An *execution* of an algorithm for ring R of size n is an infinite sequence of triples (C_1,M,C_2), where C_1 and C_2 are configurations and M is a message vector, all of size n. We require executions to satisfy several properties. First, the initial configuration must be R. Second, the second configuration in each triple must be the same as the first configuration in the next triple. Finally, each triple in an execution must describe correct message generation, as given by μ, and correct state changes, as described earlier. An *execution fragment* is any finite prefix of an execution.

We now define our complexity measures. We measure the number of messages sent and the number of rounds taken only up to the point where a processor becomes elected. (This convention only serves to

strengthen our lower bound.) For any execution e, let *finishtime(e)* denote the number of the first round after which a processor has entered a state in E. Let *messages(e)* denote the number of messages sent during e, up to and including round finishtime(e).

2.3. Election of a Leader

Let X be an ID space with $|X| \geq n$. A ring algorithm over X is said to *elect a leader in rings of size n* provided that in each execution, e, of the algorithm, for a ring R of size n over X, exactly one processor eventually enters a state in E.

2.4. Comparison Algorithms

We say that two S-expressions, s and s', over X are *order-equivalent* provided that they are structurally equivalent as S-expressions, and if two atoms from s satisfy one of the order relations $<$, $=$ or $>$, then the corresponding atoms from s' satisfy the same relation. An algorithm is a *comparison algorithm* provided that if s and s' are order-equivalent well-formed S-expressions over X, then processors with states s and s' transmit messages in the same direction or directions and have the same election status. That is, μ(s,counterclockwise) = μ(s',counterclockwise), μ(s,clockwise) = μ(s',clockwise), and s is in E exactly if s' is in E.

3. Chains

In this section, we describe the general theory needed for our lower bound proof for comparison algorithms. We introduce the concept of a "chain", which describes information flow during an execution of a ring algorithm. The notion of a "chain" used in this paper is a substantial generalization of the notion of a "chain" used for a similar purpose in [FL]. For comparison algorithms, we show that nonexistence of certain chains implies that certain processors in a ring remain indistinguishable. Our definition of a chain is rather unusual, but turns out to be exactly the right description of information flow to prove this indistinguishability.

3.1. Basic Definitions

A k-*segment* of a ring is a length k sequence of consecutive processors in the ring, in clockwise order. Let S and T be two k-segments in a ring, with first processors p and q respectively and last processors p' and q' respectively, and let e be an execution (or execution fragment) of an algorithm in the ring. Then a *clockwise chain* in e for (S,T) is a length k subsequence of the steps of e, $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$, such that the following is true. In each step e_{i_j}, a message is sent either by processor $p + j \cdot 2$ to processor $p + j \cdot 1$, or else by processor $q + j \cdot 2$ to processor $q + j \cdot 1$. Thus, a clockwise chain for a pair of segments describes combined information

flow clockwise in the two segments, from outside the two segments up to the last processors p' and q'. A *counterclockwise chain* in e for (S,T) is defined analogously, for information flow counterclockwise: in each step e_j, a message is sent either by processor p' · j + 2 to processor p' · j + 1, or else by processor q' · j + 2 to processor q' · j + 1.

Two length k vectors of X-elements are said to be *order-equivalent* provided that the elements in corresponding positions satisfy the same ordering relations in the two vectors. (That is, if the two vectors are a and b, then a_i and a_j satisfy the same relation, <, = or >, as b_i and b_j.) Two segments S and T are said to be *order-equivalent* in a particular ring R provided that the sequences of initial ID's of the processors in the two segments are order-equivalent.

Let e be an execution fragment. Then *maxcw(e)* is defined to be the maximum k for which there are order-equivalent length k segments S and T (possibly with S = T), such that e contains a clockwise chain for (S,T). The quantity *maxccw(e)* is defined analogously. Let *sum(e)* = maxcw(e) + maxccw(e).

3.2. Limitations on Chains

First, it is obvious that a length 0 execution e has maxcw(e) = maxccw(e) = sum(e) = 0.

The following lemma limits the growth of chains at a single step of an execution.

Lemma 1: Let e be an execution fragment for a ring R, and e' another execution fragment consisting of all but the last step of e. Then (a) maxcw(e) $\leq$ maxcw(e') + 1, with maxcw(e) = maxcw(e') if no messages are sent clockwise at the last step of e, and (b) maxccw(e) $\leq$ maxccw(e') + 1, with maxccw(e) = maxccw(e') if no messages are sent counterclockwise at the last step of e.

Proof: We argue part (a). Part (b) is analogous. The second half of the claim is obvious. We argue the inequality maxcw(e) $\leq$ maxcw(e') + 1. We may assume that maxcw(e) $\geq$ 1, since otherwise the result is obvious.

Let S and T be order-equivalent segments of length maxcw(e) for which there is a clockwise chain in e. Let S' and T' be the segments of length maxcw(e) - 1 consisting of all but the last processor in S and T respectively. Then S' and T' are order-equivalent. Moreover, since only the last message in the chain could have been sent at the last step of e, it must be that e' contains a clockwise chain for (S',T'). Thus, maxcw(e') $\geq$ maxcw(e) - 1, as required. ∎

3.3. Bisegments

In the next subsection, we will show that, for comparison algorithms, limitations on chains in an execution imply limitations on distinguishability of processors. In order to state those results, we require some additional definitions.

If k_1 and k_2 are positive integers, a (k_1,k_2)-*bisegment* is defined to be a pair of segments, the first of size k_1 and the second of size k_2, which overlap in a single processor (the last processor of the first segment and the first of the second segment). The processor which appears in both segments is called the *center* of the bisegment. The *spanning segment* of a bisegment is the segment obtained by concatenating the two segments in the bisegment, and removing the duplicated center. Two bisegments are said to be *order-equivalent* in a particular ring provided their spanning segments are order-equivalent. Two processors p and q are (k_1,k_2)-*equivalent* in a particular ring provided that their (k_1,k_2)-bisegments (i.e. the (k_1,k_2)-bisegments centered at p and q) are order-equivalent.

Let $S = (S_1,S_2)$ and $T = (T_1,T_2)$ be two (k_1,k_2)-bisegments, e an execution or execution fragment. Then a *clockwise chain* in e for (S,T) is a clockwise chain in e for (S_1,T_1), and a *counterclockwise chain* for (S,T) is a counterclockwise chain for (S_2,T_2). A *chain* in e for (S,T) is either a clockwise chain or a counterclockwise chain for (S,T).

3.4. Indistinguishability

In this subsection, we give a lemma which shows that, for comparison algorithms, absence of certain chains implies that certain processors must remain "indistinguishable".

Our notion of "indistinguishability" is defined as follows. If S and T are two ID sequences, each of length k, and s and t are two S-expressions, then s is *congruent* to t with respect to (S,T) provided that s and t are structurally equivalent, and corresponding positions in s and t contains elements from corresponding positions of S and T, respectively. If S and T are two segments of a particular ring, then s and t are *congruent* with respect to (S,T) provided that s and t are congruent with respect to the corresponding sequences of ID's. Similarly, if S and T are two bisegments of a ring, we say that s and t are *congruent* with respect to S and T provided that they are congruent with respect to their spanning segments.

> **Lemma 2:** Let p and q be (k_1,k_2)-equivalent processors in a ring R, and let S and T be their respective (k_1,k_2)-bisegments. Let e be an execution fragment of a comparison algorithm for R. If there are no chains in e for (S,T), then at the end of e, the states of p and q are congruent with respect to (S,T).

Proof: The proof is by induction on the length of e.

Base: $|e| = 0$. Neither p nor q has received any messages in e, so they will remain in states which are congruent with respect to (S,T).

Inductive step: $|e| > 0$, and the result holds for any execution fragment of length shorter than $|e|$ and any values of k_1 and k_2. Let e' denote e except for its last step. Then by inductive hypothesis, p and q remain in states which are congruent with respect to (S,T) up to the end of e'. Consider what happens at the last step. Let p' and q' be the respective counterclockwise neighbors of p and q, and p'' and q'' the respective clockwise neighbors.

Case 1: Both of the following hold: (a) Either p' and q' are in states which are congruent with respect to (S,T) just after e', or else neither p' nor q' sends a message clockwise at the last step of e. (b) Either p'' and q'' are in states which are congruent with respect to (S,T) just after e', or else neither p'' nor q'' sends a message counterclockwise at the last step of e.

In this case, it is easy to see that p and q remain in states which are congruent with respect to (S,T), after e. For if p' and q' are in states which are congruent with respect to (S,T) just after e', then since the algorithm is a comparison algorithm, they both make the same decision about whether or not to send a message clockwise at the last step of e. If they both send a message, Then the messages they send are just their respective states, which are congruent with respect to (S,T). A similar argument applies to p'' and q''. It follows that p and q remain in states which are congruent with respect to (S,T) after the last step of e.

Case 2: Processors p' and q' are in states which are not congruent with respect to (S,T) just after e', and at least one of them sends a message clockwise at the last step of e.

If $k_1 = 1$ (i.e. if p and q are at the counterclockwise ends of their respective bisegments), then a clockwise chain for (S,T) is produced by the message sent at the last step, a contradiction. So assume that $k_1 > 1$. Since p and q are (k_1,k_2)-equivalent, it follows that p' and q' are (k_1-1,k_2+1)-equivalent. Let S' and T' denote their respective (k_1-1,k_2+1)-bisegments. S' and T' contain exactly the same processors as S and T respectively, but are centered at p' and q' rather than p and q. Since the states of p' and q' just after e' are not congruent with respect to (S,T), they are also not congruent with respect to (S',T'). By the inductive hypothesis, there must be a chain in e' for (S',T'). If there is a counterclockwise chain in e' for (S',T'), then it is also a counterclockwise chain for (S,T), so there is a counterclockwise chain in e for (S,T). On the other hand, if there is a clockwise chain in e' for (S',T'), then since at least one of p' and q' sends a message clockwise at the last step of e, we obtain a clockwise chain in e for (S,T). Either case is a contradiction.

Case 3: Processors p'' and q'' are in states which are not congruent with respect to (S,T) just after e', and at least one of them sends a message counterclockwise at the last step of e. The argument is analogous to the one for Case 2. ∎

Thus, we have shown that absence of certain chains implies that certain processors will remain congruent.

wo corollaries which will be used in our lower bound proof follow from this lemma. The first one says that, hen chains are short and there are lots of equivalent processors, any message which gets sent has many ɔrresponding messages sent at the same time by other processors.

Corollary 3: Let k be a positive integer. Assume ring R is such that every k-segment has at least i order-equivalent k-segments. Let e be any execution fragment of a comparison algorithm in R, e' be another fragment consisting of all but the last step of e, and assume that sum(e') < k. If some processor p sends a message clockwise (or counterclockwise) at the last step of e, then there are at least i processors that do the same.

Proof: Consider the case where p sends a message clockwise. The other case is analogous. Let k_1 = maxcw(e') + 1 and k_2 = maxccw(e') + 1. The (k_1,k_2)-bisegment for p has at most k elements, so that p has at least i (k_1,k_2)-equivalent processors. Let q be any one of these processors, and let S and T be the (k_1,k_2)-bisegments centered at p and q, respectively. Then there cannot be a chain in e' for (S,T), by the definitions of maxcw and maxccw. But then Lemma 2 implies that p and q remain congruent with respect to (S,T) at the end of e'; since the algorithm is a comparison algorithm, q also sends a message clockwise at the last step of e. ∎

ɜmma 2 also has the following consequence for comparison algorithms to elect a leader. This corollary says at long chains must be generated in order to elect a leader, if certain equivalent processors exist.

Corollary 4: Let k be a positive integer. Let R be a ring in which every k-segment S has another order-equivalent k-segment T. Let e be any execution fragment of a comparison algorithm which elects a leader in R, such that a leader gets elected in e. Then sum(e) $\geq$ k.

Proof: Assume not - that sum(e) = maxcw(e) + maxccw(e) < k. Let k_1 = maxcw(e) + 1 and k_2 = maxccw(e) + 1. The (k_1,k_2)-bisegment for the processor p that gets elected leader has at most k elements, so that p has a (k_1,k_2)-equivalent processor q ≠ p; let S and T be the (k_1,k_2)-bisegments centered at p and q, respectively. Then there cannot be a chain in e for (S,T), by the definition of maxcw and maxccw. But then Lemma 2 implies that p and q remain congruent with respect to (S,T); since the algorithm is a comparison algorithm, p and q cannot be distinguished as to leadership. This is a contradiction. ∎

. Replication Symmetry in Rings

n this section, we show, for each n, how to construct a ring, R_n, consisting of n ID's chosen from a ırticular ID space. The ring R_n is constructed to have a large amount of replication symmetry. That is, gments in R_n have many order-equivalent segments. In the next section, we will show that this replication mmetry causes the R_n to require a large number of messages for election of a leader.

1. Hierarchical Organization of Processors

Fix a particular ring size $n \geq 1$. R_n will be constructed by first constructing a sequential pattern P_n and then ınnecting its ends to form a ring. The construction of P_n groups the processors using a hierarchy of depth where $d = \lfloor (\log_9 n)/2 \rfloor$. We describe the grouping using a derivation tree of a context-free grammar. ıter, we will use the structure of the derivation tree to assign ID's to the n leaves of the tree and thereby oduce pattern P_n.

Define the context-free grammar G as follows. The nonterminals, representing groups of processors, are A_i ıd B_i, $1 \leq i \leq d$, plus B_0. There is just one terminal symbol, p, representing a processor. The start symbol is). The productions are:

$B_i \rightarrow B_{i+1}A_{i+1}A_{i+1}B_{i+1}B_{i+1}A_{i+1}A_{i+1}B_{i+1}B_{i+1}$, for $0 \leq i \leq d-1$,

$A_i \rightarrow A_{i+1}B_{i+1}B_{i+1}A_{i+1}A_{i+1}B_{i+1}B_{i+1}A_{i+1}A_{i+1}$, for $1 \leq i \leq d-1$,

$B_d \rightarrow p^{b_d}$, and $A_d \rightarrow p^{a_d}$.

(That is, B_d generates a string consisting of b_d p symbols, and analogously for A_d.) The quantities a_d and b_d ill be defined later, in such a way as to guarantee that the length of the unique sentence generated by G is n.

For each i, $0 \leq i \leq d$, define the *level i sentential form* of G to be the unique string over $\{A_i,B_i\}$ derivable in . There are exactly 9^i nonterminal symbols in the level i sentential form. Moreover, for each i, the number of ymbols A_i is exactly one less that the number of symbols B_i.

Lemma 5: In the level i sentential form of G, $0 \leq i \leq d$, the number of symbols A_i is $\lfloor 9^i/2 \rfloor$, and the number of symbols B_i is $\lceil 9^i/2 \rceil$.

Proof: By induction on i. ∎

All A_i nodes derive a terminal string of the same length; let us call this length a_i. Similarly, all B_i nodes derive terminal string of the same length, which we will call b_i. Let $c_i = \min(a_i,b_i)$, for all i, $1 \leq i \leq d$.

We next describe how to select the values a_d and b_d. They are chosen in such a way that the total length of ıe unique sentence derived in G is exactly n, and so that $|b_d - a_d|$ is small. We use the following.

Lemma 6: Let m, $n \geq 0$ be integers. Then there are integers a and b such that $n = am + b(m + 1)$ and $|b - a| \leq m$.

Proof: Fix m. If $m = 0$, then $a = b = n$ suffices, so assume that $m \geq 1$. We proceed by induction on n.

Basis: $n = 0$. Then $a = b = 0$ suffices.

Inductive step: Assume that $n = am + b(m + 1)$ and $|b - a| \leq m$. We will produce a' and b' such that $n + 1 = a'm + b'(m + 1)$ and $|b' - a'| \leq m$. There are two cases:

Case 1. $b - a \leq m - 2$. Then let $a' = a - 1$ and $b' = b + 1$. The equation is satisfied, and $b' - a' = b - a + 2$. Then $b' - a' \geq b - a \geq -m$, and $b' - a' \leq (m - 2) + 2 = m$, as needed.

Case 2. $b - a \geq m - 1$.

Then let $a' = a + m$ and $b' = b - m + 1$. The equation is satisfied, and $b' - a' = b - a - 2m + 1$. Then $b' - a' \geq m - 1 - 2m + 1 = -m$, and $b' - a' \leq b - a$ since $m \geq 1$. Thus, $b' - a' \leq m$, as needed. ∎

Let $m = \lfloor 9^d/2 \rfloor$. It is easy to see that m is $\Theta(n^{1/2})$, and in particular, that $m \leq n^{1/2}/2$. Using Lemma 6, choose a_d and b_d to be integers such that $n = a_d m + b_d(m + 1)$ and $|b_d - a_d| \leq m$. We must show that a_d and b_d are nonnegative: if either of a_d and b_d is negative, then $\max(a_d, b_d) \leq m - 1$, so $n = a_d m + b_d(m + 1) \leq 2(m^2) \leq n/2$, a contradiction.

Lemma 7: The length of the unique sentence generated by G is n.

Proof: By Lemma 5, there are exactly $\lfloor 9^d/2 \rfloor = m$ symbols A_d, and exactly $\lceil 9^d/2 \rceil = m + 1$ symbols B_d *in the level d sentential form of G. Since* $n = a_d m + b_d(m + 1)$, the result holds. ∎

We have already noted that m is $\Theta(n^{1/2})$. Since a_d is nonnegative, we have that $n \geq b_d(m+1)$. Using the lower bound on m, we see that b_d is $O(n^{1/2})$.

The final lemma of this subsection gives the exact value of the difference $c_i - c_{i+1}$.

Lemma 8: The difference $c_i - c_{i+1} = 4 \cdot 9^{d-(i+1)}(n - b_d)/m$, for $0 \leq i \leq d - 1$.

Proof: We first show that if $a_{i+1} < b_{i+1}$, then $a_i < b_i$. This follows since $a_i = 5a_{i+1} + 4b_{i+1} < 4a_{i+1} + 5b_{i+1} = b_i$. A similar fact holds if $b_{i+1} < a_{i+1}$. Next we note that $a_i - a_{i+1} = b_i - b_{i+1} = 4(a_{i+1} + b_{i+1})$. Thus no matter which of a_{i+1} and b_{i+1} is smaller, $c_i - c_{i+1} = 4(a_{i+1} + b_{i+1})$.

From the choice of a_d and b_d, we have $a_d + b_d = (n - b_d)/m$. It follows that $a_{i+1} + b_{i+1} = 9^{d-(i+1)}(n - b_d)/m$. Substituting into the expression for $c_i - c_{i+1}$ gives the desired result. ∎

4.2. Labelling of Processors

Let X be the ID space consisting of all strings of length $d + 1$ whose elements are nonnegative integers, with the strings ordered lexicographically; X is the ID space from which the pattern P_n will be constructed.

We define P_n by describing an assignment of ID's to n processors, corresponding to the leaves of the

derivation tree of G. In order to do this, we associate *labels* with the nodes of the derivation tree. The label of the root of the tree is the null string. If a node with a corresponding nonterminal A_i or B_i, $0 \leq i \leq d - 1$, is labelled by the string w, then the labels of its nine children are respectively w0, w1, w2, w3, w8, w7, w6, w5, w4. If a node with a corresponding nonterminal A_d is labelled by the string w, then the labels of its a_d children are respectively w0, w1,..., w(a_d - 1). If a node with a corresponding nonterminal B_d is labelled by the string w, then the labels of its b_d children are respectively w0, w1,..., w(b_d - 1). Processor ID's are generated by interpreting the labels of the leaves as elements of X, i.e. as length d + 1 strings of nonnegative integers, ordered lexicograpically.

In the level i sentential form of G, define an ordered pair of nonterminal symbols to be "of type A>A" provided that it consists of the two symbols A_iA_i, and the label of the node of the first nonterminal is lexicographically greater than that of the second. We use analogous definitions for types A<A, A>B, A<B, B>A, B<A, B>B and B<B.

We now show that the level i sentential form has equal numbers of consecutive pairs of nonterminals of the eight possible types.

Lemma 9: In the level i sentential form of G, $0 \leq i \leq d$, the number of occurrences of consecutive pairs of each of the eight types A>A, A<A, A>B, A<B, B>A, B<A, B>B and B<B is exactly $\lfloor 9^i/8 \rfloor$.

Proof: It suffices to show that the numbers of occurrences of the eight types of pairs are equal, since the total number of pairs is exactly $9^i - 1 = 8 \lfloor 9^i/8 \rfloor$. We proceed by induction on i. For the basis, i = 0, the result is vacuously true. Assume that the result is true for i, and consider the level i + 1 sentential form. There are two kinds of pairs of level i + 1 nonterminals: those in which both elements are derived from the same level i nonterminal node, and those in which the two elements are derived from two different level i nonterminal nodes. Each level i nonterminal node generates a length 9 sequence of level i + 1 nonterminals, in which each of the eight types of pairs has exactly one occurrence. Therefore, there are equal numbers of the eight possible types among the pairs which are derived from the same level i nonterminal node. Also, each pair which is derived from two different level i nonterminal nodes is of the same type as the corresponding pair of parent nodes; the inductive hypothesis implies that there are equal numbers of the eight possible types among these pairs, as well. The result follows. ∎

In any level i sentential form, note that the pair consisting of the last nonterminal node followed by the first nonterminal node, is of type B>B.

4.3. Replication Symmetry

The ring R_n is constructed by folding the sequential pattern P_n constructed in the preceding subsections into a ring; the sequential pattern P_n is arranged around the circle in clockwise order. In this subsection, we state a lemma which describes the replication symmetry of R_n. This lemma will be used in the next section, to yield our lower bound for the number of messages required by a comparison algorithm to elect a leader.

Lemma 10: Let $1 \leq i \leq d$. Let S be any segment of R_n of length at most $c_i + 1$. Then there are at least $\lfloor 9^i/8 \rfloor$ segments of R_n that are order-equivalent to S, including S itself.

Proof: Let S be a segment of R_n, of length at most $c_i + 1$. Then S is contained in the subtrees of at most two nonterminal nodes at level i. These two are either two consecutive nonterminal nodes, or else the last and first nonterminals in the sentential form. Let t be the type of this ordered pair of nonterminal nodes.

By Lemma 9, there are at least $\lfloor 9^i/8 \rfloor$ instances of type t consecutive pairs of nonterminal nodes in the level i sentential form. Each of these instances of a pair of type t contains a segment which is order-equivalent to S.

∎

5. Lower Bound

In this section, we state and prove our lower bound for the number of messages required by a comparison algorithm to elect a leader. We use the symmetric rings R_n constructed in the previous section, and the two corollaries from Section 3.

Theorem 11: Let $\mathcal{A}$ be a comparison algorithm over an arbitrary ID space, X, which elects a leader in rings of size n. Then there is an execution, e, of $\mathcal{A}$ for which messages(e) $\geq \Omega(n \log n)$.

Proof: Assume n is fixed, and at least 9^4. This ensures that the depth $d = \lfloor (\log_9 n)/2 \rfloor$ is at least 2. It suffices to consider the ID space X consisting of length d + 1 strings of nonnegative integers, ordered lexicographically. Assume the ring R_n is used. Let e be the execution fragment for R_n, which terminates just when the elected processor enters an "elected" state. By Lemma 10, every segment of length $c_2 + 1$ has at least one other order-equivalent segment in R_n. (The Lemma actually implies that there are at least nine others, but we do not require this fact here.) Thus, by Corollary 4, execution e must progress from having a sum of 0 to having a sum of at least $c_2 + 1$.

Consider any step of e at which the sum first stops being at most k, for any $k \leq c_i$. By Lemma 9, the sum increases by at most 2 at this step. Moreover, if no messages are sent clockwise (resp., counterclockwise) at this step, then the sum increases by at most 1.

Let e' be the prefix of e preceding this step. Then sum(e') $< c_i + 1$. Lemma 10 implies that any segment of length $c_i + 1$ has at least $\lfloor 9^i/8 \rfloor$ order-equivalent segments in R_n. Thus by Corollary 3, if any messages are sent clockwise at this step, then at least $\lfloor 9^i/8 \rfloor$ messages are sent clockwise, and similarly for messages sent counterclockwise. Thus, if the sum increases by 1 at this step, at least $\lfloor 9^i/8 \rfloor$ messages are sent, while if the sum increases by 2 at this step, then at

least twice that number of messages are sent. It follows that the cost of increasing the sum from 0 to at least $c_2 + 1$ can be apportioned as a cost of at least $\lfloor 9^i/8 \rfloor$ for each increase from k to k + 1, where $k \leq c_i$.

We now total up the number of messages sent in e. Grouping increases according to level, we see that the number of messages sent must be at least

$$\Sigma_{i=2,\ldots,d-1} \lfloor 9^i/8 \rfloor (c_i - c_{i+1}).$$

By Lemma 8, this quantity is equal to

$$\Sigma_{i=2,\ldots,d-1} \lfloor 9^i/8 \rfloor (4 \cdot 9^{d-(i+1)} (n - b_d)/m)$$

$$= 4((n - b_d)/m) \Sigma_{i=2,\ldots,d-1} \lfloor 9^i/8 \rfloor 9^{d-(i+1)}$$

$$\geq 4((n - b_d)/m) [\Sigma_{i=2,\ldots,d-1} 9^i/8\, 9^{d-(i+1)} - \Sigma_{i=2,\ldots,d-1} 9^{d-(i+1)}].$$

The first summation evaluates to $(d-2)\, 9^{d-1}/8$, while the second is bounded above by $9^{d-2}/8$. Thus, the message bound is at least

$$4((n - b_d)/m) [(d-2)\, 9^{d-1}/8 - 9^{d-2}/8].$$

Since $m = \lfloor 9^d/2 \rfloor \leq 9^d/2$, this is at least

$$8((n - b_d)/9^d) [(d-2)\, 9^{d-1}/8 - 9^{d-2}/8]$$

$$= (n - b_d) [(d-2)/9 - 1/81] = (n - b_d) [d/9 - O(1)].$$

Since b_d is $O(n^{1/2})$, the message bound is at least

$$= (n - O(n^{1/2})) [d/9 - O(1)]$$

$$= (n - O(n^{1/2})) [((1/2)\log_9 n)/9 - O(1)]$$

$$= n((1/2)\log_9 n)/9 - O(n)$$

$$= (n \log_2 n)/(18 \log_2 9) - O(n). \blacksquare$$

We have just shown that algorithms which are restricted to use only comparisons of ID's require at least $\Omega(n \log n)$ messages in the worst case. We now state a corollary which shows that the given lower bound applies for time-bounded algorithms as well. More specifically, we conclude that there is a (very fast-growing) function f with the following property. If the time is required to be bounded by some t in the worst case, and

ID's are chosen from any ID space having at least f(n,t) elements, then any algorithm requires $\Omega(n \log n)$ messages in the worst case.

Corollary 12: There exists a function f such that for any n, t, the following holds. Let X be an arbitrary ID space with at least f(n,t) elements. Let $\mathcal{A}$ be any algorithm over X which elects a leader in rings of size n, and for which finishtime(e) $\leq$ t for all executions e for rings of size n. Then there is an execution, e, of $\mathcal{A}$ for which messages(e) is $\Omega(n \log n)$.

Proof: The proof is analogous to a similar one in [FL]. It is based on a transformation from time-bounded algorithms to comparison algorithms, using Ramsey's theorem. (The idea for the transformation, presented in the paracomputer model, originally appeared in [S1]. Snir [S2] credits Yao [Y] with inspiration for the construction.) ∎

6. Remaining Questions

The general $\Omega(n \log n)$ bound which we have proved has a very small constant, $1/(18 \log_2(9))$. In contrast, the best constant known for an upper bound is around 1.4 [P,DKR]. It remains to close this gap. For certain values of n (such as powers of 2), we do have a narrower gap, because the proof in [FL] produces a lower bound with a larger constant. It is possible that there are certain properties of the number n (e.g. properties of its prime factorization) that affect the size of the constant. It would be interesting to understand these relationships.

Acknowledgements:

The authors thank Maria Klawe for her interest and encouragement in our attempts to obtain rings with replication symmetry, and Mark Tuttle for his comments on early versions of the manuscript.

References:

[B] J. E. Burns
A formal model for message passing systems,
TR-91, Indiana University (September 1980).

[CR] E. Chang and R. Roberts
An improved algorithm for decentralized extrema-finding in circular configurations of processes,
1979 281-283.

[DKR] D. Dolev, M. Klawe and M. Rodeh
An *O (n* log *n)* unidirectional distributed algorithm for extrema finding in a circle,
J. Algorithms 3,3 (September 1982) 245-260.

[FL] G. Frederickson and N. Lynch
The impact of synchronous communication on the problem of electing a leader in a ring,
Proceedings of the 16th Annual ACM Symposium on Theory of Computing,
Washington, D.C.,

(April 30 - May 2, 1984), 493-503.

[GHS] R. Gallager, P. Humblet and P. Spira
A distributed algorithm for minimum-weight spanning trees,
ACM Transactions on Programming Languages and Systems,
Vol. 5, No. 1, (January 1983), 66-77.

[HS] D. S. Hirschberg and J. B. Sinclair
Decentralized extrema-finding in circular configurations of processes,
Comm. ACM 23 (November 1980), 627-628.

[IR] A. Itai and M. Rodeh
The lord of the ring or probabilistic methods for breaking symmetry in distributive networks, IBM Research Report RJ3110, (4/3/81).

[I] G. LeLann
Distributed systems - toward a formal approach,
Information Processing 77, North Holland, Amsterdam
(1977) 155-160.

[P] G. L. Peterson
An *O (n* log *n)* unidirectional algorithm for the circular extrema problem,
Trans. Prog. Lang. Sys. 4, 4 (1982) 758-762.

[S1] M. Snir
On parallel searching, Hebrew University of Jerusalem,
Department of Computer Science, RR 83-21 (June 1983).

[S2] M. Snir,
Personal communication (1983).

[V] P. Vitanyi,
Distributed elections in an archimedean ring of processors,
Proceedings of the 16th Annual ACM Symposium on Theory of Computing,
Washington, D. C., (April 30 - May 2, 1984), 542-547.

[Y] A. Yao,
Should tables be sorted? *J. ACM 28*, 3 (July 1981) 615-628.

OFFICIAL DISTRIBUTION LIST

1985

Director
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

2 Copies

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. R. Grafton, Code 433

2 Copies

Director, Code 2627
Naval Research Laboratory
Washington, DC 20375

6 Copies

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314

12 Copies

National Science Foundation
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

2 Copies

Dr. E.B. Royce, Code 38
Head, Research Department
Naval Weapons Center
China Lake, CA 93555

1 Copy

Dr. G. Hopper, USNR
NAVDAC-OCII
Department of the Navy
Washington, DC 20374

1 Copy

END

FILMED

8-85

DTIC