ACT-50

SEPTEMBER, 1984

CSL COORDINATED SCIENCE LABORATORY

APPLIED COMPUTATION THEORY GROUP

LOWER BOUNDS ON COMMON KNOWLEDGE IN DISTRIBUTED ALGORITHMS

E. GAFNI
M.C. LOUI
P. TIWARI
D.B. WEST
S. ZAKS

REPORT R-1017

UILU-ENG 84-2212

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) R-Report #1017, ACT-50 (UILU-ENG-84-2212)	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Laboratory University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Joint Services Electronics Program (JSEP) National Science Foundation (NSF)
6c. ADDRESS (City, State and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State and ZIP Code) 800 N. Quincy St. 1800 G St. N.W. Arlington, VA 22217 Washington, D.C. 20550 (JSEP) (NSF)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Prog. National Science Foundation 8b. OFFICE SYMBOL (If applicable) N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER (JSEP) N00014-79-C-0424 (NSF) MCS-8217445
8c. ADDRESS (City, State and ZIP Code) 800 N. Quincy St.,Arlington,VA 22217 (ONR) 1800 G St. N.W.,Washington,D.C. 20550 (NSF)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: N/A PROJECT NO.: N/A TASK NO.: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) Lower Bounds on Common Knowledge...	
12. PERSONAL AUTHOR(S) E. Gafni, M.C. Loui, P. Tiwari, D.B. West, S. Zaks	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT (Yr., Mo., Day) September 1984 15. PAGE COUNT 21
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Distributed algorithms, communication complexity

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We establish lower bounds on the communication complexity of several distributed algorithms that achieve common knowledge. On a ring of N processors every comparison algorithm that solves the plurality problem or the distinctness problem requires $\Omega(N^2)$ messages. On a ring of N processors every algorithm that solves the distinctness problem requires $\Omega(N^2 \log (L/N))$ bits among its messages. We include precise definitions of distributed algorithms and their executions.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) 22c. OFFICE SYMBOL NONE

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

LOWER BOUNDS ON COMMON KNOWLEDGE IN DISTRIBUTED ALGORITHMS*

Eliezer Gafni[1]

Michael C. Loui[2]

Prasoon Tiwari[3]

Douglas B. West[4]

Shmuel Zaks[5]

September 1984

Abstract

We establish lower bounds on the communication complexity of several distributed algorithms that achieve common knowledge. On a ring of N processors every comparison algorithm that solves the plurality problem or the distinctness problem requires $\Omega(N^2)$ messages. On a ring of N processors every algorithm that solves the distinctness problem requires $\Omega(N^2 \log (L/N))$ bits among its messages. We include precise definitions of distributed algorithms and their executions.

*This research was performed at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.

[1]Department of Computer Science, University of California at Los Angeles, Los Angeles, California 90024. Supported by the Joint Services Electronics Program (U.S. Air Force, U.S. Army, U.S. Navy) under Contract N00014-79-C0424.

[2]Department of Electrical and Computer Engineering and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. Supported by the National Science Foundation under Grant MCS-8217445 and by the Eastman Kodak Company.

[3]Department of Electrical and Computer Engineering and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. Supported by the National Science Foundation under Grant MCS-8217445.

[4]Department of Mathematics, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

[5]Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. Supported by the National Science Foundation under Grant MCS-8302391. On leave from the Department of Computer Science, Technion, Haifa, Israel.

1. INTRODUCTION

An algorithm for a distributed computer system achieves common knowledge if it computes a function that requires the participation of all processors. Korach et al. (1984) call such a function global. The processors compute this global function by exchanging some local information at each step.

Efficient distributed algorithms have been designed to compute maxima (Dolev et al., 1982; Peterson, 1982), medians (Frederickson, 1983; Rodeh, 1982; Santoro and Sidney, 1982), minimum spanning trees (Gallager et al., 1983), shortest paths (Chandy and Misra, 1982), and maximum flows (Segall, 1982). Each of these algorithms achieves common knowledge.

In this paper we study further problems on distributed systems. We establish new lower bounds on the communication complexity of activation, plurality, and distinctness problems. These problems are ostensibly simpler than the problems previously investigated, yet they are fundamental: algorithms that achieve common knowledge often involve decisions about activation of processors or distinctness of input values. Thus we believe that our techniques will yield lower bounds when applied to other problems.

The communication complexity of an algorithm is measured by the number of messages or the number of bits that are transmitted on communication links by the processors executing the algorithm. Several lower bounds on communication complexity are known. As usual, to express these lower bounds, $\Omega(g(N))$ denotes a function f such that for some constant c, $f(N) \geq c\, g(N)$ for all N sufficiently large. For the election problem Burns (1980) obtained a lower bound of $\Omega(N \log N)$ messages in the worst case on a bidirectional ring with N processors. Frederickson and Lynch (1984) derived an $\Omega(N \log N)$ lower bound for election even when the ring is synchronous. Pachl et al. (1982) demonstrated that $\Omega(N \log N)$ messages are necessary on the average for election on a unidirectional ring. For the sorting problem Loui (1983) proved that when the values are in $\{0,\ldots,L\}$, every algorithm requires $\Omega(N^2 \log (L/N))$ bits among messages on a bidirectional ring. For the computation of minimum spanning trees Santoro (1982) and Korach et al. (1984) established $\Omega(N \log N)$ lower bounds on messages on various networks with N processors.

Section 2 defines precisely the execution of a distributed algorithm and the two performance measures: message complexity and bit complexity. Also, this Section defines the problems that we discuss. Section 3 presents some elementary results, including the message complexity of the activation problem. Section 4 treats the message complexities of the distinctness and plurality problems. Section 5 discusses the bit complexity of the distinctness problem.

2. DEFINITIONS

2.1. The Computational Model

We adopt the model of asynchronous distributed computation developed by Santoro (1981, 1982). After describing the model informally, we give complete, precise definitions.

A distributed system comprises identical processors connected via a communication network. Processor y can send a message directly to processor z if and only if link (y,z) is in the network. The transmission of a message incurs an unpredictable but finite delay. Messages sent on the same link (y,z) arrive at z in the same order as they were sent.

Every processor executes the same algorithm, which specifies the messages sent by the processor. Any message transmitted by a processor depends only on the sequence of messages that it has received. Initially, each processor knows only the links that involve it; thus the algorithm cannot use information about the global structure of the network. Each processor y has an identifier ID(y) and an initial value V(y). The processors exchange messages to compute a function of these values. At the end of the computation, every processor y has a result R(y).

In a bidirectional ring, each processor can exchange messages only with its two neighbors. To each processor in a bidirectional ring assign an integer p, $0 \leq p < N$. If integer p is assigned to processor y and integer q is assigned to processor z, then we shall refer to "processor p" and to "link (p,q)." Thus the bidirectional ring has links (p, p − 1 mod N) and (p, p + 1 mod N) for all p. The assignment of integers to processors is used only for clarity of exposition; since the processors are identical, processor p does not have access to the number p. Although we derive lower bounds on bidirectional rings, our techniques could be applied to networks with other topologies.

The message complexity of an algorithm is a function that assigns to every N the maximum of the number of messages used by the algorithm on distributed systems with N processors. The bit complexity of an algorithm is a function that assigns to every N the maximum of the number of bits in all messages used by the algorithm on distributed systems with N processors. Abelson (1980), Ja' Ja' and Kumar (1984), Papadimitiou and Sipser (1984), and Yao (1979) studied the bit complexity measure in similar contexts.

Let us define the computational model precisely. A distributed system is an octuple (PROC, LINKS, MES, IDEN, VAL, RES, In, Out), where

PROC is a finite set of processors,
LINKS $\subseteq$ PROC x PROC is a set of links,
MES is a set of messages,
IDEN is a set of identifiers,
VAL is a set of initial values,
RES is a set of results, and
In and Out are functions defined below.

For simplicity assume that every processor y has the same number d of incoming links of the form (w_i, y) and the same number d of outgoing links of the form (y, z_i). At each processor assign to each incoming link a distinct number in $\{1, ..., d\}$. The function

$$\text{In: LINKS} \rightarrow \{1, ..., d\}$$

expresses these assignments. At each processor assign to each outgoing link a distinct number in $\{1, ..., d\}$. The function

$$\text{Out: LINKS} \rightarrow \{1, ..., d\}$$

expresses these assignments.

An initial value distribution is a function $y \rightarrow V(y)$ from PROC to VAL. An identifier distribution is a function $y \rightarrow ID(y)$ from PROC to IDEN. A result distribution is a function $y \rightarrow R(y)$ from PROC to RES.

The computation by each processor depends only on its identifier, its initial value, and the sequence of messages that it has received. We formalize this notion.

An <u>event</u> is the transmission or arrival of a message at a processor y. An event is specified by listing the processor, the message, the link number, and whether it is a transmission or an arrival.

Each processor has a current state. The <u>state</u> of a processor y comprises an identifier id, an initial value v, and a sequence of zero or more events at y. Thus a state has the form

$$\langle id, v, e_1, e_2, \ldots, e_k \rangle,$$

where $e_1, e_2, \ldots, e_k$ are events. Call k the <u>length</u> of the state. State s' is a <u>successor</u> of state s if s is a prefix of s'. In response to an event e, a processor in state s undergoes a transition into a new state that results from the concatenation of e onto the end of s. Let STATES be the set of states.

Each link (y,z) has a finite queue Q(y,z) of messages. A message can be enqueued onto the rear of Q(y,z) or dequeued from the front of Q(y,z).

A <u>configuration</u> is a function C that specifies states for the processors and message queues for the links. If y is a processor, then C(y) is a state. If (y,z) is a link, then C(y,z) is a queue of messages. A configuration C_0 is <u>initial</u> if the length of every state $C_0(y)$ is 0 and every queue $C_0(y,z)$ is empty.

A <u>distributed algorithm</u> is a function

$$A: \text{STATES} \rightarrow \{\emptyset\} \cup (\text{MES} \times \{1,\ldots,d\}) \cup \text{RES}$$

that specifies what a processor does in any state. If processor y is in state s, then either y does nothing ($A(s) = \emptyset$); or y transmits a message on an outgoing link, as specified by $A(s) \in \text{MES} \times \{1,\ldots,d\}$; or y concludes with a result $A(s) \in \text{RES}$. If $A(s) \in \text{MES} \times \{1,\ldots,d\}$, then A(s) induces a transmission event at y. A state s is <u>terminal</u> for A if for every successor s' of s, including s itself, A(s') is a fixed result. In other words, if processor y is in a terminal state, then despite state transitions caused by arrival events, y neither transmits messages nor changes its result.

An <u>execution</u> of an algorithm A is a finite sequence of configurations

$$C_0, C_1, C_2, \ldots, C_t,$$

starting from an initial configuration C_0, such that for every i, C_{i+1} is obtained from C_i in one of two ways:

(1) concatenating a transmission event e_i induced by $A(C(y_i))$ at some processor y_i onto the end of the state of y_i, and enqueuing the corresponding message onto the link (y_i,z_i) indicated by e_i; or

(2) concatenating an arrival event e_i onto the end of the state of some processor z_i, and dequeueing the corresponding message from the link (y_i,z_i) indicated by e_i.

An execution <u>terminates</u> if in its last configuration C_t all processor states are terminal and all message queues are empty.

A <u>problem</u> is a function P that maps an initial configuration C_0 to a result distribution $P(C_0)$. In the sequel we shall consider restrictions of problems to bidirectional rings; in these cases we shall assume that P is defined only for initial configurations on bidirectional rings. An algorithm A <u>solves</u> a problem P if for every initial configuration C_0 every execution of A starting from C_0 terminates, and the results computed by A agree with the result distribution $P(C_0)$.

These definitions ensure the properties of distributed algorithms described earlier in this Section. The system is asynchronous; an algorithm may have many executions for the same identifier distribution and initial value distribution. All messages sent on the same link arrive reliably in the order in which they were sent. Every processor executes the same algorithm. Finally, the behavior of a processor depends only on the messages that it has received. Since the algorithm uses numbers in $\{1,\ldots,d\}$ to identify links, it cannot take advantage of the size of the system.

If a terminating execution has t events, then it has exactly t/2 transmission events and t/2 arrival events because all message queues become empty. For brevity we shall say that the execution has "t/2 messages." The <u>message complexity</u> of an algorithm A is a function g(N) that assigns to each N the maximum number of messages in executions of A on a distributed system of N processors. To define the bit complexity assume that MES is a prefix-free collection of binary strings; these binary strings encode the actual messages. The prefix-free property enables the receiving processor to parse a sequence of messages. The <u>bit complexity</u> of an algorithm A is a function that assigns to each N the maximum number of bits among messages in executions of A on a distributed system of N processors.

Our lower bounds on bit complexity apply to all algorithms. To obtain lower bounds on message complexity we consider comparison algorithms, which we define below. Our definitions resemble the definitions of Frederickson and Lynch (1984), who studied synchronous systems.

Assume that VAL $\subseteq$ MES and that VAL is a totally ordered set. Furthermore, assume that two elements in MES are comparable only if both are in VAL, or if both are not in VAL; the elements in VAL are incomparable with elements not in VAL. Use the symbol $<$ for the ordering relation on MES. For a state s and $i \geq 1$ define $M_i(s)$ to be the message in the ith event in s. Define $M_{-1}(s)$ to be the initial identifier in s and $M_0(s)$ to be the initial value in s. States s and s' are <u>order-equivalent</u> if and only if

(1) they have the same length k;

(2) for all $-1 \leq i, j \leq k$, $M_i(s)$ and $M_j(s)$ are related in the same way as $M_i(s')$ and $M_j(s')$ — that is, in both cases the relation is $<$ or $=$ or $>$ or incomparable; and

(3) for every j the jth events in s and s' are either transmission events on the same outgoing link or arrival events on the same incoming link.

An algorithm A is a <u>comparison</u> <u>algorithm</u> if whenever s and s′ are order-equivalent states, the following conditions hold:

(1) A(s) and A(s′) belong to the same set among ϕ, MES $\times$ {1,...,d}, RES;

(2) if A(s) $\in$ RES, then A(s) = A(s′); in particular, if s is terminal, then s′ is terminal;

(3) if A(s) $\in$ MES $\times$ {1,...,d}, then the successor states that result from the transmission events induced by A(s) and A(s′) are order-equivalent -- that is, the messages transmitted from states s and s′ preserve the order-equivalence;

(4) if A(s) specifies a message m $\notin$ IDEN $\cup$ VAL, then A(s′) specifies the same message m;

(5) if A(s) specifies a message m $\in$ IDEN $\cup$ VAL, then m is either the identifier in s or the initial value in s or a message in one of the events in s.

2.2. Problems

1. Activation: Notify every processor that every processor in the system is active. An algorithm solves the Activation Problem if in all executions of the algorithm, each processor y enters a terminal state only after every processor has transmitted a message. For this problem the initial values are irrelevant.

2. Election: Elect exactly one leader in a network. An algorithm solves the Election Problem if there is exactly one processor w such that the result at w is R(w) = "elected," and the result at y is R(y) = ID(w) for all $y \neq w$. Call processor w the leader. For this problem the initial values are irrelevant.

3. Plurality: Determine the plurality value in a network with arbitrary initial values. An algorithm solves the Plurality Problem if the result at every processor y is the same R(y) = v, where v is an initial value that appears at least as often as every other value among the initial values.

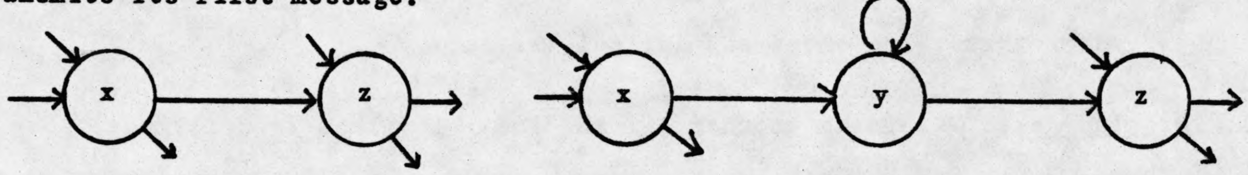
4. Distinctness: Determine whether all the initial values are distinct. For every processor y the result R(y) = 1 if all values in the initial value distribution are distinct, and R(y) = 0 otherwise.

3. ELEMENTARY RESULTS

<u>Theorem 1.</u> An algorithm that solves the Activation, Election, Plurality, or Distinctness Problem on every system uses at least e messages when the system has e links.

<u>Proof</u>. Suppose, to the contrary, an algorithm that solves one of these problems uses fewer than e messages for all initial configurations on a system S with e links. It follows that for every terminating execution of the algorithm, there is a link (x,z) on which no messages are transmitted.

For this execution S is indistinguishable from a system S′ that has an additional processor y with links (x,y) and (y,z), but no link (x,z). Thus for the execution of the algorithm on S′, processors x and z reach terminal states before y transmits its first message.



S   S′

For the Activation Problem, z has already reached a terminal state before y transmits a message, a contradiction. For the Election Problem, since x sends no messages, y will not receive the identifier of the leader, a contradiction.

For the Plurality and Distinctness Problems, although y may send messages to z, the result at z remains the same. But varying the initial value at y could change the results of these problems. []

We prove that unless the processors have distinct identifiers, the Election Problem cannot be solved by a distributed algorithm.

<u>Theorem 2.</u> There is no algorithm that solves the Election Problem in networks that permit multiple copies of identifiers, even if the pattern of identifiers is asymmetric.

Proof. Suppose, to the contrary, algorithm A uses at most M messages to elect a leader on a bidirectional ring S with N processors. For any identifier distribution on S, splice together $2 \lceil \frac{M}{N} \rceil + 2$ copies of this identifier distribution to form a linear array of $(2 \lceil \frac{M}{N} \rceil + 2)N$ processors. Join the ends of this array to a new processor z with a new identifier. Call the resulting bidirectional ring S'. In S' there are adjacent copies S_1, S_2 of S such that every processor in S_1 or S_2 is at least distance M from z. Since no messages that originate at z affect processors in S_1 or S_2, both S_1 and S_2 will elect leaders. []

Angluin (1980) gave a similar proof for her Theorem 4.6. Henceforth we assume that the processors have distinct identifiers.

Theorem 3. Let N be a power of 2. On a bidirectional ring of N processors every comparison algorithm that solves the Activation Problem has message complexity $\Omega(N \log N)$.

To establish Theorem 3, we first define a chain. If during an execution of an algorithm processor y_1 sends a message to another processor y_2, then the sequence

$$(y_1, y_2)$$

is a chain of length 1. In general, a chain of length k is a sequence of distinct processors

$$(y_1, \ldots, y_{k-1}, y_k, y_{k+1})$$

such that $(y_1, \ldots, y_{k-1}, y_k)$ is a chain of length k-1, and y_k sends a message to y_{k+1} after the chain message from y_{k-1} arrives at y_k.

We shall modify the proof of Frederickson and Lynch (1984) for the Election Problem when N is a power of 2. Frederickson and Lynch considered a particular identifier distribution that has some symmetry properties. They showed that for every execution of any algorithm that solves the Election Problem on a ring with this identifier distribution, the length of some chain is at least N/2.

Furthermore, because of the symmetry in the identifier distribution, every comparison algorithm requires at least $\frac{N}{2} \log_2 N$ messages, in the worst case, to establish a chain of length N/2.

Proof of Theorem 3. Let A be a comparison algorithm that solves the Activation Problem on a ring of N processors. Assume the initial configuration specifies the identifier distribution of Frederickson and Lynch (1984). We shall demonstrate that for every execution of A there is a chain whose length is at least N/2.

Suppose, to the contrary, in some execution of A the length of the longest chain is less than N/2. Let y be a particular processor, and let Y be the set of processors z such that in this execution z is in a chain that ends at y. Let Y′ be the set of processors not in Y. By hypothesis, Y′ is not empty: the processor diametrically opposite from y is in Y′.

Construct another execution of A, starting from the same initial configuration, in which no processor in Y′ sends messages. In this new execution y receives the same sequence of messages as before and enters the same terminal state before every processor has transmitted a message. This is a contradiction.

Now the argument of Frederickson and Lynch implies the desired $\Omega(N \log N)$ lower bound on the message complexity of A. []

4. THE MESSAGE COMPLEXITY OF THE DISTINCTNESS AND PLURALITY PROBLEMS

An algorithm could solve the Distinctness Problem on a ring of N processors with O(N log N) messages: (1) use O(N log N) messages to elect a leader; (2) use N messages of increasing length to accumulate all the initial values and deliver them to the leader; and (3) after the leader decides whether the initial values are distinct, use N messages to deliver the result to the other processors. A comparison algorithm must use $\Omega(N^2)$ messages, however. The proof of this fact must overcome several subtleties.

Theorem 4. On a bidirectional ring of N processors every comparison algorithm that solves the Distinctness Problem has message complexity $\Omega(N^2)$.

Proof. Let A be a comparison algorithm that solves the Distinctness Problem. We describe a collection of distributions for which in the worst case some execution of A has $\Omega(N^2)$ messages.

Let VAL be a set of N values $\{v_0, v_1, \ldots, v_{N-1}\}$ such that $v_i < v_j$ if and only if $i < j$. Let D be the collection of distributions V for which

$$V(p) \in \{v_{2p}, v_{2p+1}\} \text{ for } p = 0, \ldots, N/2 - 1,$$

$$V(p + N/2) \in \{v_{2p}, v_{2p+1}\} \text{ for } p = 0, \ldots, N/2 - 1,$$

Fix any distribution V in D for which

$$V(p) \neq V(p + N/2) \quad \text{for all } p.$$

One might surmise that to solve the Distinctness Problem each diametrically opposite pair of values V(q) and V(q + N/2) must be compared. Motivated by this intuition, we shall demonstrate that for every processor q, at the end of every terminating execution of A on V some processor state s must contain both V(q) and V(q + N/2). It follows that the values V(q) and V(q + N/2) themselves must have been transmitted as messages N/2 times because under A, a processor y may transmit a value v only if v is in the state of y. Since A uses N/2 messages for every q = 0, ..., N/2 - 1, it uses at least $(N/2)(N/2) = \Omega(N^2)$ messages.

Let

$$C_0, C_1, \ldots, C_t$$

be a terminating execution of A, and suppose that for some q no $C_i(y)$ contains both $V(q)$ and $V(q + N/2)$. We shall derive a contradiction. Let V′ be the distribution defined by

$$V'(p) = V(p) \text{ for all } p \neq q,$$

$$V'(q) = V'(q + N/2) = V(q + N/2);$$

whereas all values of V are distinct, not all values of V′ are distinct. Since A is a comparison algorithm, the computation of A on V′ should resemble its computation on V. More precisely, define C'_0 to be the initial configuration in which V′ is the initial value distribution. We shall construct inductively a terminating execution

$$C'_0, C'_1, \ldots, C'_t$$

of A on V′ that satisfies the following <u>Substitution Property</u> for each k:

every $C'_k(y)$ will differ from $C_k(y)$ only by the substitution of $V'(p)$ for $V(p)$ for all p;

every $C'_k(y,z)$ will differ from $C_k(y,z)$ only by the substitution of $V'(p)$ for $V(p)$ for all p.

By hypothesis, since $C_k(y)$ does not have both $V(q)$ and $V(q + N/2)$, the Substitution Property implies that $C_k(y)$ is order-equivalent to $C'_k(y)$. By definition of C'_0, the Substitution Property holds for $k = 0$.

Suppose

$$C'_0, C'_1, \ldots, C'_k$$

have been defined. Consider the event e_k associated with the transition from C_k to C_{k+1}. There are two possibilities. First, suppose e_k is an arrival event on link (y_k, z_k). Define C'_{k+1} to be the configuration induced by an arrival event on (y_k, z_k) from configuration C'_k. Since the Substitution Property holds for C'_k, it holds for C'_{k+1} too. Second, suppose e_k is a transmission event on link (y_k, z_k), and let m_k be the message sent by y_k in state $C_k(y_k)$. Because $C_k(y_k)$ is order-equivalent to $C'_k(y_k)$, processor y_k can also send a message m'_k on (y_k, z_k) in state $C'_k(y_k)$. Let C'_{k+1} be the configuration that results from C'_k by the transmission of m'_k on (y_k, z_k). Since A is a comparison algorithm, the state $C_{k+1}(y_k)$, whose last event has m_k, is order-equivalent to the state $C'_{k+1}(y_k)$, whose last event has m'_k. Thus, if $m_k \in$ IDEN $\cup$ VAL and m_k is the message in the jth event in $C_k(y_k)$, then m'_k is the message in the jth event in $C'_k(y_k)$; it follows by the Substitution Property for C'_k that if $m_k = V(p)$ for some p, then $m'_k = V'(p)$, hence the Substitution Property holds for C'_{k+1}. If $m_k \notin$ IDEN $\cup$ VAL, then $m_k = m'_k$, and again the Substitution Property holds for C'_{k+1}.

For every y, since $C_t(y)$ is order-equivalent to $C'_t(y)$ and A is a comparison algorithm, the results of the two executions are the same: for every y,

$$A(C_t(y)) = A(C'_t(y)).$$

But because the values specified by V are distinct, $A(C_t(y)) = 1$; and because the values specified by V' are not distinct, $A(C'_t(y)) = 0$. Contradiction! We have shown that for some k and some y*, $C_k(y^*)$ has both $V(q)$ and $V(q + N/2)$. []

Whereas Frederickson and Lynch (1984) use order-equivalent states during the same execution, this proof involves order-equivalent states in two different executions.

The Distinctness Problem reduces to the Plurality Problem. After finding a plurality value v, an algorithm on a ring can use O(N) more messages to determine whether v occurs more than once among the initial values. Thus the lower bound of Theorem 4 implies the same lower bound for the the Plurality Problem.

Corollary. On a bidirectional ring of N processors every comparison algorithm that solves the Plurality Problem has message complexity $\Omega(N^2)$.

5. THE BIT COMPLEXITY OF THE DISTINCTNESS PROBLEM

Although comparison algorithms permit arbitrary messages -- not just initial values, they seem weak. A comparison algorithm cannot achieve a small message complexity by encoding several initial values into one compact message: the initial values themselves must be transmitted. Unrestricted algorithms might solve the Distinctness Problem more efficiently. For example, let the initial values VAL = $\{0,\ldots,L\}$. To solve the Distinctness Problem, an algorithm could arrange the initial values into sorted order; then it could check whether two adjacent values in this order are equal. To sort the initial values $O(N^2 \log (L/N))$ bits suffice (Loui, 1983). Our next lower bound asserts that $\Omega(N^2 \log (L/N))$ bits are necessary.

<u>Theorem 5.</u> If $L \geq N$, then on a bidirectional ring of N processors with initial values in $\{0,\ldots,L\}$, every algorithm that solves the Distinctness Problem has bit complexity $\Omega(N^2 \log (L/N))$.

To prove Theorem 5 we shall use a technique developed by Tiwari (1984). This technique generalizes the results of Mehlhorn and Schmidt (1982), who studied systems with just two processors.

In a distributed system S partition the processors into sets $Y_0, \ldots, Y_k$ such that every link joins processors in Y_i and Y_{i+1} for some i. Let w_i be the number of links between Y_i and Y_{i+1}, and let $w = \max_i [w_i]$. Consider the computation of a binary function f(U,V) on S, where U is the set of initial values in Y_0 and V is the set of initial values in Y_k; the initial values in $Y_1, \ldots, Y_{k-1}$ are irrelevant. At termination the result at every processor is f(U,V). Define the <u>results</u> <u>matrix</u> R for f: the rows and columns of R are indexed by sets of initial values, and for each U, V,

$$R_{UV} = f(U,V).$$

Let rank(R) denote the rank of R.

<u>Lemma 1</u> (Tiwari, 1984). On S the bit complexity of every algorithm that computes f is

$$\Omega(k \frac{\log (\mathrm{rank}(R))}{1 + \log w}).$$

The results matrix R to which we shall apply Lemma 1 has a special structure. To construct R we shall use a function F on matrices defined as follows: if B is a b x b matrix, then

$$F(r,B) = \begin{bmatrix} 0 & B & B & \dots & B \\ B & 0 & B & \dots & B \\ B & B & 0 & \dots & B \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ B & B & B & \dots & 0 \end{bmatrix}$$

is a br x br matrix, where 0 denotes a constant b x b matrix whose entries are all zero. F(r,B) has r^2 blocks, each of which is either 0 or B.

<u>Lemma 2.</u> If B is nonsingular and $r \geq 2$, then F(r,B) is nonsingular.

<u>Proof</u>. Suppose there are vectors $x_1, \dots, x_r$, each with b components, such that

$$F(r,B) \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_r \end{bmatrix} = 0$$

By definition of F,

$$B \sum_{i \neq j} x_i = 0 \quad \text{for all } j = 1, \dots, r. \tag{1}$$

Sum Equation (1) over all j:

$$B \sum_j \sum_{i \neq j} x_i = (r - 1)\, B \sum_j x_j = 0.$$

Since B is nonsingular and $r \geq 2$,

$$\sum_j x_j = 0. \tag{2}$$

By (1) and (2),

$$B\,(-x_j) = 0 \quad \text{for all } j = 1, \ldots, r,$$

hence since B is nonsingular, every $x_j = 0$. []

An alternative proof of Lemma 2 follows from the observation that $F(r,B)$ is a Kronecker product of two matrices. Let J_r be the constant $r \times r$ matrix whose entries are all 1, and let I_r be the $r \times r$ identity matrix. Then

$$F(r,B) = (J_r - I_r) \otimes B.$$

The determinant of the Kronecker product of two matrices is a product of powers of their determinants:

$$\det F(r,B) = (\det(J_r - I_r))(\det B)^r = (r-1)(-1)^{r-1}(\det B)^r \neq 0$$

because B is nonsingular. Therefore $F(r,B)$ is nonsingular.

<u>Proof of Theorem 5.</u> Let $r = 2\lfloor L/N \rfloor$. Since $L \geq N$, $r \geq 2$. Define a collection of initial value distributions V by

$$\{V(p),\ V(p + N/2)\} \subseteq \{pr,\ pr+1,\ \ldots,\ (p+1)r - 1\}$$
$$\text{for } p = 0, \ldots, N/2 - 1$$

such that $V(p) \neq V(p + N/2)$ for $p = N/4, \ldots, N/2 - 1$. The initial values are distinct if and only if $V(p) \neq V(p + N/2)$ for all $p = 0, \ldots, N/4 - 1$.

Partition the bidirectional ring into $N/4 + 2$ sets of processors as follows:

Y_0 = {processors $0, 1, \ldots, N/4 - 1$},

Y_1 = {processors $N/4, N-1$},

Y_2 = {processors $N/4 + 1, N-2$}, ...,

$Y_{N/4}$ = {processors $N/2 - 1, 3N/4$},

$Y_{N/4+1}$ = {processors $N/2, N/2 + 1, \ldots, 3N/4 - 1$}.

Let $V(Y_0)$ be the set of initial values at Y_0 and $V(Y_{N/4+1})$ be the set of initial values at $Y_{N/4+1}$. The initial values specified by V are distinct if and only if no integer is in both $V(Y_0)$ and $V(Y_{N/4+1})$. For sets U and U′ of initial values define a function f by

$$f(U,U') = \begin{cases} 0 \text{ if some integer is in both } U \text{ and } U' \\ 1 \text{ if no integer is in both } U \text{ and } U'. \end{cases}$$

Any algorithm that solves the Distinctness Problem computes $f(V(Y_0),V(Y_{N/4+1}))$.

Using the function F defined above, we determine the results matrix for f. Let $B_0 = [1]$, a 1 x 1 matrix, and for n = 1, ..., N/4 let

$$B_n = F(r,B_{n-1}).$$

The $r^{N/4} \times r^{N/4}$ matrix $R = B_{N/4}$ is the results matrix for the function f. Each row of R corresponds to a set of initial values for Y_0, each column to a set of initial values for $Y_{N/4+1}$.

Now we apply Lemma 1. For our partition k = N/4 + 1. Since between every Y_i and Y_{i+1} there are exactly 4 links, w = 4. By induction and Lemma 2, R is nonsingular; consequently $\text{rank}(R) = r^{N/4}$. Thus by Lemma 1 the bit complexity of any algorithm that computes f is

$$\Omega((N/4 + 1) \frac{\log (r^{N/4})}{1 + \log 4}) = \Omega(N^2 \log r) = \Omega(N^2 \log (L/N)). \quad []$$

REFERENCES

Abelson, H. (1980), Lower bounds on information transfer in distributed systems, J. Asso. Comput. Mach. 27, 384-392.

Angluin, D. (1980), Local and global properties in networks of processors, in "Proc. 12th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 82-93.

Burns, J.E. (1980), "A formal model for message passing systems," Tech. Rep. 91, Comput. Sci. Dept., Indiana Univ. at Bloomington, May 1980.

Chandy, K.M., and Misra, J. (1982), Distributed computation on graphs: Shortest path algorithms, Commun. Asso. Comput. Mach. 25, 833-837.

Dolev, D., Klawe, M., and Rodeh, M. (1982), An O(n log n) unidirectional distributed algorithm for extrema finding in circles, J. Algorithms 3, 245-260.

Frederickson, G.N. (1983), Tradeoffs for selection in distributed networks, in "Proc. 2nd ACM Symp. on Principles of Distributed Computing," Association for Computing Machinery, New York, 154-160.

Frederickson, G.N., and Lynch, N.A. (1984), The impact of synchronous communication on the problem of electing a leader in a ring, in "Proc. 16th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 493-503.

Gallager, R.G., Humblet, P.A., and Spira, P.M. (1983), A distributed algorithm for minimum-weight spanning trees, ACM Trans. Prog. Lang. Syst. 5, 66-77.

Ja' Ja', J., and Kumar, V.K.P. (1984), Information transfer in distributed computing with applications to VLSI, J. Assoc. Comput. Mach. 31, 150-162.

Korach, E., Moran, S., and Zaks, S. (1984), Tight lower and upper bounds for some distributed algorithms for a complete network of processors, in "Proc. 3rd ACM Symp. on Principles of Distributed Computing," Association for Computing Machinery, New York, to appear.

Loui, M.C. (1983), "The complexity of sorting on distributed systems," Tech. Rep. R-995 (ACT-39), Coordinated Science Lab., Univ. Illinois at Urbana-Champaign. To appear in Inform. Control.

Mehlhorn, K., and Schmidt, E.M. (1982), Las Vegas is better than determinism in VLSI and distributed computing, in "Proc. 14th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, pp. 330-337.

Papadimitriou, C.H., and Sipser, M. (1984), Communication complexity, J. Comput. System Sci. 28, 260-269.

Peterson, G.L. (1982), An O(n log n) unidirectional algorithm for the circular extrema problem, ACM Trans. Prog. Lang. Syst. 4, 758-762.

Pachl, J., Korach, E., and Rotem, D. (1982), A technique for proving lower bounds for distributed maximum-finding algorithms, in "Proc. 14th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 378-382.

Rodeh, M. (1982), Finding the median distributively, J. Comput. Syst. Sci. 24, 162-166.

Santoro, N. (1981), Distributed algorithms for very large distributed environments: New results and research directions, in "Proc. Canad. Inform. Processing Soc.," Waterloo, 1.4.1 - 1.4.5.

Santoro, N. (1982), "On the message complexity of distributed problems," Tech. Rep. SCS-TR-13, School of Computer Science, Carleton Univ., Dec. 1982.

Santoro, N., and Sidney, J.B. (1982), Order statistics on distributed sets, in "Proc. 20th Ann. Allerton Conf. on Communication, Control, and Computing," Univ. Illinois at Urbana-Champaign, 251-256.

Segall, A. (1982), Decentralized maximum-flow protocols, Networks 12, 213-220.

Tiwari, P. (1984), Lower bounds on communication complexity in distributed computer networks, in "Proc. 25th Ann. IEEE Symp. on Foundations of Computer Science," Institute of Electrical and Electronics Engineers, New York, to appear.

Yao, A.C.C. (1979), Some complexity questions related to distributive computing, in "Proc. 11th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 209-213.