Symmetry breaking in distributive networks

Conference Paper *in* Foundations of Computer Science, 1975., 16th Annual Symposium on · November 1981

DOI: 10.1109/SFCS.1981.41 · Source: IEEE Xplore

2 authors:

Alon Itai
Technion - Israel Institute of Technology
119 PUBLICATIONS **7,134** CITATIONS
SEE PROFILE

Michael Rodeh
IBM
78 PUBLICATIONS **3,310** CITATIONS
SEE PROFILE

SYMMETRY BREAKING IN DISTRIBUTED NETWORKS

Alon Itai
Computer Science Department
Technion - Israel Institute of Technology
Haifa, Israel

Michael Rodeh
IBM Israel Scientific Center
Technion City
Haifa, Israel

ABSTRACT

Given a ring of n processors it is required to design the processors such that they will be able to choose a leader (a uniquely designated processor) by sending messages along the ring. If the processors are indistinguishable then there exists no deterministic algorithm to solve the problem. To overcome this difficulty, probabilistic algorithms are proposed. The algorithms may run forever but they terminate within finite time on the average.

For the synchronous case several algorithms are presented: The simplest requires, on the average, the transmission of no more than $2.442n$ bits and $O(n)$ time. More sophisticated algorithms trade time for communication complexity. If the processors work asynchronously then on the average $O(n\log n)$ bits are transmitted.

In the above cases the size of the ring was assumed to be known to all the processors. If the size is not known then finding it may be done only with high probability: any algorithm may yield incorrect results (with nonzero probability) for some values of n. Another difficulty is that, if we insist on correctness, the processors may not explicitly terminate. Rather, the entire ring reaches an inactive state, in which no processor initiates communication.

1. INTRODUCTION

Given a network of *n* processors it is required to program the processors to solve some network problems, such as adding the numbers stored in the local memory of the processors or finding their maximum. This task is easier if the processors are distinguishable (i.e., by having unique names). However, if all processors are identical then the problem becomes symmetric and solving global network problems becomes harder.

For a concrete example, we consider a *ring,* a cycle of *n* indistinguishable processors (i.e., with no *id*'s), and discuss two global problems:

(i) Choosing a *leader* — a uniquely designated processor, such that each processor knows whether it is the leader;

(ii) Finding *n* — the size of the ring.

Angluin [A] has investigated the problem of choosing a leader using Milne and Milner's [MM] model for distributed systems (her results may be applied to other models as well). She has shown that there exists no single program which runs on all the vertices of a ring of arbitrary size and designates a unique leader within finite time. The fundamental phenomenon is that symmetry cannot be broken without allowing either an infinite computation or an erroneous result. Global information (such as knowing the size of the ring) does not always help.

Since either termination or correctness must be compromised, we shall construct probabilistic algorithms [R1] — algorithms which usually terminate with a correct result. Thus, we assume that even though the processors are identical, each has an independent random number generator. (For a use of probabilistic algorithms for symmetry breaking in a shared memory environment, see [R2].) Our theme is that to break symmetry one can develop a probabilistic routine to suggest and improve solutions together with a routine to test the correctness of the proposed solution. Obviously, we are interested in efficient algorithms, whose *communication complexity* — the number of bits transmitted — is low.

The problem of choosing a leader when *n* is known is an example of a case where correctness is achievable. This is proven in Sections 2 and 3: Section 2 discusses the synchronous case while in Section 3 algorithms for the asynchronous case are presented. The problem of choosing a leader is easier in asymmetric networks. For example, if each processor has a unique *id* then the processor with the maximum *id* may serve as the leader ([Bu], [CR], [DKR], [F1], [HS], [L], [Pe]).

The problem of finding *n*, the size of the ring, is discussed in Section 4. This problem is directly related to that of termination. Usually when a processor has computed the required value it is aware of this fact and terminates. An algorithm *processor terminates* when all processors have terminated. However, all activities may cease if all the processors reach a state from which they will not initiate any communication unless receiving some message and there are no pending messages in the system. This is the weaker notion of *message termination*. An external observer may, for example, detect message termination by examining all links. In Section 4 we show that these notions are distinct: If a value of N is known for which $N \leq n < 2N$ for some N then there exists an algorithm to find *n* which processor terminates (Section 4.2). Otherwise, there exists no terminating algorithm with bounded probability of error (Section 4.1). We must, therefore, resort to the weaker notion of message termination. In Section 4.3 we develop an algorithm, for which within polynomial time, all communications cease and with high probability the computed result is correct.

Some of the results of this paper appeared in [IR]. Here besides correcting minor errors, we have reformulated the computational model and through the selection/verification paradigm given a more unifying approach to all the algorithms. This paper also contains an improved algorithm to choose a leader in an asynchronous ring (section 3.4) and an algorithm to find *n* when possible (section 4.3.1).

Following [IR], Abrahamson et al. [AAHK] considered the subproblem of solitude verification – verifying that there is only one live processor. They gave upper and lower bounds for asynchronous rings where *n* is known, and in rings where a good estimate for *n* exists. Their paper considered only message termination and considered the probability of error. Duris and Galil [DG] have shown a lower bound on the average time required for finding the maximum in a ring (the average is taken over all distributions of the input). Pachl [Pa] and Itai et al [IMN] give a lower bound over all randomized algorithms.

2. CHOOSING A LEADER IN A SYNCHRONOUS RING

2.1. The Model

A *unidirectional ring* consists of n processors $v_0, \ldots, v_{n-1}$ connected by directed edges $(v_i, v_{i+1})^{\dagger}$. Every processor v has a (possibly infinite) set of states. The network is *synchronous* if time is divided into an infinite number 0,1,2,... of *time slots.* The processors start executing at time slot $t=0$. At every time slot t a processor v_i reads a message that was sent to it at time slot $t-1$ if such a message exits, makes exactly one state transition (i.e., perform an arbitrary computation) and may send at most one message to v_{i+1}. The new state of v_i and the message sent depend on both the previous state and the message sent by v_{i-1} in the previous time slot.

An *algorithm* for a ring is the state diagram of all the processors. At any time slot t the state of the network is completely determined by the state of each processor and the messages sent during time slot $t-1$. Each processor has a *halting state* — a state from which all transitions lead back to the same state without generating any messages. The algorithm *terminates* when all the processors arrive at a halting state. The *time complexity* of a synchronous algorithm is the number of time slots until the algorithm terminates and the *message complexity* (*bit complexity*) is the total number of messages (bits) sent by all the processors during the entire run of the algorithm.

To add randomization we assume that each processor v has an infinite sequence $\{r[v,t]\}_{t=1}^{\infty}$ of real numbers, $0 \le r[v,t] \le 1$, and that the t-th transition of v depends also on $r[v,t]$. The message complexity of an algorithm A is then a function of r and is denoted by $C(A,r)$. If for all v and t the values of $r[v,t]$ are drawn with uniform probability then $\overline{C}(A)$ is the average over all r's. The reason for choosing random real numbers instead of random bits is that our algorithms sometimes perform actions with probability $1/n$ and unless n is a power of 2 this cannot be simulated with a finite number of bits.

The aim of this paper is to study the effect of symmetry; thus we have restricted our attention to a network which is topologically symmetric and whose processors are indistinguishable, i.e., have the same state diagram. In particular, they do not have unique *id*'s. Each processor v may, however, have an independent random number generator which is modeled by the aforementioned infinite sequence $\{r[v,t]\}_{t=1}^{\infty}$. In this section we assume that the processors know n, the size of the ring. Therefore, for each

† All computations on indices are done modulo n – the size of the ring.

n we may have a different algorithm.

2.2. Choosing a leader of a ring

We show first that a leader may be chosen within $O(n)$ average bit complexity. The exact constant depends on the amount of memory of each processor. In this subsection the processors use only $\log n$ bits to record the choices of the other processors. In the next subsection more local memory is required, but the bit complexity is reduced.

The algorithm proceeds in phases, each of n time slots. At every phase, $a \le n$ processors are *active.* During a phase, some of the active processors may become inactive, eventually reducing a to 1.

At the beginning of a phase, every active processor v decides with probability a^{-1} whether to become a *candidate.* At the end of the phase every processor has calculated c — the number of candidates of this phase. If $c=1$ then the sole candidate becomes the leader. If $c>1$ then the active processors of the next phase are the candidates of the current phase. Finally, if $c=0$ then the phase was *useless,* all active processors remain active and their number is not reduced.

At time 0 all processors become active and start the algorithm. Hence, initially $a=n$. To compute c, each candidate sends a pebble at the beginning of the phase. This pebble is passed around the ring, returning at the end of the phase (after exactly n time slots) to its originator. Every processor deduces c by counting the number of pebbles which passed through.

Program no. 1 below gives the details of the program for processor v.

Let $p(a,c)$ denote the probability that c out of a active processors chose to become candidates. Since each one of the a active processors chose to become a candidate with probability a^{-1},

$$p(a,c) = \binom{a}{c} a^{-c} \left(1-\frac{1}{a}\right)^{a-c} = \left(1-\frac{1}{a}\right)^{a} \binom{a}{c} (a-1)^{-c}.$$

Theorem 2.1: Let $l(a)$ denote the expected number of phases required to reduce the number of active processors from a to one. Then $l(a)$ converges to 2.441716... .

Proof: We first calculate l:

$$l(a) = 1 + p(a,0)\, l(a) + \sum_{k=1}^{a} p(a,k)\, l(k).$$

```
begin
{ Initialization }
        $time := 0$:
        $active := candidate :=$ true;
        $a := c := n$;
        case $time$ of
        (i) $time$ is not divisible by $n$:
                if there is a pebble in the buffer then
                        begin send it on; $c := c+1$  end
        (ii) $time$ is divisible by $n$: {including the case $time = 0$}
                if $c=1$ then if $candidate$ then $terminate$('I am the leader')
                                        else $terminate$('I am not the leader')
                else begin
                        if $c>1$ then begin  $a := c$;  $active := candidate$ end;
                        {the following is executed for both $c=0$ and $c>1$}
                        if $active$ and $r[v, time] \le a^{-1}$ then
                                begin $candidate :=$ true; $c := 1$; send a pebble end
                        else  begin $candidate :=$ false;  $c := 0$ end
                end
end.
```

Program no. 1

By definition, $l(1) = 0$. Thus,

$$l(a)(1-p(a,0)-p(a,a)) = 1+\sum_{k=2}^{a-1} p(a,k)l(k)$$

$$l(a) = \frac{1+\sum_{k=2}^{a-1} p(a,k)l(k)}{1-p(a,0)-p(a,a)} \tag{2.1}$$

We now make several observations:

Claim 1: $l(a)<e$.

Proof: Induction on a.

Basis: $l(1)=0$.

Induction step: By (2.1) and the induction hypothesis,

$$l(a) \le \frac{1+e(1-p(a,0)-p(a,1)-p(a,a))}{(1-p(a,0)-p(a,a))}$$

$$= e + \frac{1-p(a,1)e}{(1-p(a,0)-p(a,a))} < e.$$

The last inequality holds since $p(a,1) = \left(1-\frac{1}{a}\right)^a \frac{a}{a-1}$ is a monotonic decreasing sequence which converges to e^{-1}, hence $p(a,1)e > 1$.

Claim 2: $\lim\limits_{a\to\infty} p(a,c) = (e\cdot c!)^{-1}$.

Proof: $p(a,c)$ is a product of two factors $\left(1-\frac{1}{a}\right)^a$ and $\binom{a}{c}(a-1)^{-c}$. The first converges to e^{-1} and for fixed c the second converges to $(c!)^{-1}$. □

Let $d(a) = 1-p(a,0)-p(a,a)$. Then, $l(a) = \left(1+\sum\limits_{k=2}^{a-1} p(a,k)l(k)\right)/d(a)$.

Claim 3: $d(a)$ converges to $d(\infty) = 1-e^{-1}$.

Proof: By definition $d(a) = 1-p(a,0)-p(a,a)$. $p(a,0)=\left(1-\frac{1}{a}\right)^a$ converges to e^{-1} and $p(a,a)=a^{-a}$ converges to 0. □

Claim 4: For $a\geq 2$, $d(a)\geq ½$.

Proof: $d(2)=½$ by inspection. For $a\geq 3$, since $(1-\frac{1}{a})^a$ monotonically increases to e^{-1}, $d(a)=1-\left(1-\frac{1}{a}\right)^a-\left(\frac{1}{a}\right)^a > 1-e^{-1}-(⅓)^3 > ½$. □

Let $L(a,c) = \left(1+\sum\limits_{k=2}^{c} p(a,k)l(k)\right)/d(a)$.

Claim 5: For fixed c, $L(a,c)$ converges as a function of a.

Proof: By Claim 2, $p(a,k)$ converges. By Claim 3 so does $d(a)$, and by Claim 4, $d(a)$ is bounded away from 0. □

Let $L(\infty,c) = \lim\limits_{a\to\infty} L(a,c)$.

Claim 6: For $3\leq c<a$, $p(a,c)\leq e^{-1}/c!$.

Proof : $p(a,c) = \left(1-\frac{1}{a}\right)^a\binom{a}{c}(a-1)^{-c} < e^{-1}\frac{a(a-1)\cdots(a-c+1)}{(a-1)^c}\cdot\frac{1}{c!} < e^{-1}\frac{a(a-2)}{(a-1)^2}\cdot\frac{1}{c!} < \frac{e^{-1}}{c!}$ □

Claim 7: For $c<a$, $0<l(a)-L(a,c) < 4/c!$

Proof: By definition, $l(a)-L(a,c) = \sum\limits_{k=c+1}^{a-1} p(a,k)l(k)/d(a) > 0$. (The sum is positive since all the terms are positive.) By Claim 1, $l(k)<e$. By Claim 4, $1/d(a)\leq 2$, therefore,

$$l(a)-L(a,c) < 2e\sum_{k=c+1}^{a-1} p(a,k)$$

by Claim 6, $l(a)-L(a,c) < 2e\sum\limits_{k=c+1}^{\infty}\frac{e^{-1}}{k!} = 2\sum\limits_{k=c+1}^{\infty}\frac{1}{k!} < \frac{2}{c!}\sum\limits_{k=1}^{\infty}\frac{1}{k!} = \frac{2}{c!}(e-1) < \frac{4}{c!}$. □

Claim 8: For all c there exists a_c such that for all $a \geq a_c$

$$|l(a) - L(\infty,c)| < 8/c\,!$$

Proof: By Claim 5, for sufficiently large a $|L(a,c) - L(\infty,c)| < \frac{4}{c\,!}$. By Claim 7, $|l(a) - L(a,c)| < \frac{4}{c\,!}$.

Thus, $|l(a) - L(\infty,c)| \leq |l(a) - L(a,c)| + |L(a,c) - L(\infty,c)| < \frac{4}{c\,!} + \frac{4}{c\,!} = \frac{8}{c\,!}$. □

Claim 9: $L(\infty,c)$ is a converging sequence of c.

Proof: By Cauchy's convergence criterion it suffices to show that for all $\varepsilon > 0$ there exists a c such that if $c_1, c_2 \geq c$ $|L(\infty,c_1) - L(\infty,c_2)| < \varepsilon$.

Choose c such that $\frac{16}{c\,!} < \varepsilon$. Let $c_1, c_2 > c$. For sufficiently large a, Claim 8 shows that

$$|l(a) - L(\infty,c_1)| < \frac{8}{c_1!}$$

$$|l(a) - L(\infty,c_2)| < \frac{8}{c_2!}$$

$$|L(\infty,c_1) - L(\infty,c_2)| = |L(\infty,c_1) - l(a) + l(a) - L(\infty,c_2)| \leq |L(\infty,c_1) - l(a)| + |l(a) - L(\infty,c_2)|$$

$$< \frac{16}{c\,!} < \varepsilon.$$ □

Let L be the limit of $L(\infty,c)$ as c tends to infinity, then by Claim 8,

Claim 10: $l(a)$ converges and its limit is L.

Claim 11: $|L - L(\infty,c)| < 4/c\,!$

Proof:

$$\begin{array}{lcl} L(\infty,c) & = & \left(1 + \sum_{k=2}^{c} p(\infty,k)\, l(k)\right)/d(\infty). \\ L & = & \left(1 + \sum_{k=2}^{\infty} p(\infty,k)\, l(k)\right)/d(\infty). \\ L - L(\infty,c) & = & \left(1 + \sum_{k=c+1}^{\infty} p(\infty,k)\, l(k)\right)/d(\infty) \end{array}$$

Therefore, $L - L(\infty,c) \geq 0$. By Claim 3, $d(\infty) = 1 - e^{-1} > ½$. Using Claims 1 and 2 we get

$$0 \leq |L - L(\infty,c)| < \frac{e}{1/2} \sum_{k=c+1}^{\infty} \frac{e^{-1}}{k\,!} < 2(e-1)/c\,! < 4/c\,!$$ □

By Claim 11, $|L - L(\infty,12)| < 4/12! < 10^{-7}$. A simple calculation yields $L(\infty,12) \approx 2.441716$. Thus completing the proof of Theorem 2.1 □

Combining these results yields:

Corollary 2.2: A ring of n synchronous processors which know n may choose a leader distributedly in Ln time on the average, where $L \approx 2.441716$. The average bit complexity is Ln.

Proof: When a processors are active, a processor v sends a pebble only if $r[v, time] \le a^{-1}$. Therefore, the probability that a processor sends a pebble is a^{-1} and the average number of pebbles sent per phase is

$$\sum_{k=0}^{a} kp(a,k) = a \cdot a^{-1} = 1.$$

Thus, the expected bit complexity per phase is n. The corollary follows since the average number of phases converges to L and each phase requires n time slots. □

2.4. Better algorithms to choose a leader.

In the above algorithm the processors only counted the number of candidates. Here we follow a suggestion of Lempel [Lem] and consider an algorithm which uses additional information available to the processors. This algorithm exhibits a tradeoff between time and communication complexity by introducing a parameter q known to all the processors. As before, the algorithm proceeds in phases of length n. Prior to termination *all* the processors are active and start a phase by sending a pebble with probability q/n. To every processor v we assign an n bit vector $W(v)$ as follows: $W_i(v)=1$ iff a pebble passed through v at the i-th time slot of the phase. The $W(v)$'s at different processors v are circular shifts of one another and every processor can calculate the W of all the other processors. If all the $W(v)$'s are distinct then the leader is chosen to be the vertex whose $W(v)$ is lexicographically minimum. Otherwise, the algorithm is rerun. Notice that all the processors remain active until a leader is chosen. We state without proof:

Theorem 2.2 [IR]: In the improved algorithm the expected number of phases converges to $(1-e^{-q})^{-1}$ and the expected bit complexity per processor converges to $q(1-e^{-q})^{-1}$. □

As for the space complexity, each processor needs $O(n)$ bits of space to compute the lexicographically maximal word. However, the expected number of pebbles generated in a phase is relatively small. Thus, one can keep track only of the consecutive zeroes and thereby reduce the space complexity. By allowing at most d pebbles (and otherwise declaring a phase useless) the time and bit complexity are increased only slightly, while the space complexity is reduced to $O(d \log n)$ bits.

The above results exhibit the tradeoff between time and communication: In order to reduce the communication, more time must be spent. For $q=1$, the expected number of phases is $1\cdot(1-1/e)^{-1}\approx1.582$, and this is also the communication cost per processor. By decreasing q the expected number of phases increases to infinity while the bit complexity decreases to 1 per processor.

3. CHOOSING A LEADER IN AN ASYNCHRONOUS RING

3.1. The model

Previously we assumed that there is a global clock which governs the actions of all processors and communication. We now assume that no such clock exists, instead, each processor has its own clock which has no relation to that of the other processors. The main issue is the distinction between randomization and the nondeterminism introduced by the asynchrony of the processors. We shall try to be brief and not too formal at the expense of omitting straightforward generalizations.

Since there is no global clock we can no longer say that a message sent from v_i arrives at v_{i+1} at the next time slot. We, therefore, associate with each directed edge a buffer of unbounded size. A message sent by v_{i-1} enters the buffer associated with the edge (v_{i-1},v_i). The processor v_i waits until the longest waiting message is released by the buffer (v_{i-1},v_i). Then the processor changes its state and may send at most one message to v_{i+1}.

We may add the computation time of v_i to the time the outgoing message is held in the buffer (v_i,v_{i+1}); thus we assume that state transitions are instantaneous while the time a message stays in a buffer is unbounded and does not depend on the states of the processors. At any instance the state of the network is completely determined by the state of each processor and the contents of the buffers.

Initially, some buffers contain a *wake-up* message. The arrival of that message or any other message at a processor activates the computation.

The activity of the network is event driven by the arrival of messages. We assume that at any time at most one message arrives. The order of arrivals is governed by a *schedule* which we define to be a sequence S of edges in which each edge appears infinitely often. The schedule determines which buffer sends a message to its target processor. The requirement that every edge appears in S infinitely often implies that every message sent eventually arrives, thereby guaranteeing a property closely related to weak channel fairness as defined in [F2].

An *algorithm* A for an asynchronous ring is the state diagram of all the processors. Given a ring G, schedule S and set W of wakeup messages, the execution of the algorithm is completely determined. The *message complexity* $C(A,G,S,W)$ of A is the total number of messages it takes for A to terminate (if A does not terminate then $C(A,G,S,W)=\infty$). Also, $C(A,G) = \sup_{S,W} C(A,G,S,W)$.

Randomization is introduced, as in Section 2, by infinite sequences $\{r[v,t]\}_{t=1}^{\infty}$ $(0 \le r[v,t] \le 1)$. The t-th transition of v depends also on $r[v,t]$. The message complexity of algorithm A is then $C(A,G,S,W,r)$. If the $r[v,t]$ are drawn with uniform probability then $\overline{C}(A,G,S,W)$ is the average over all r's. (This is well defined since the behavior of A at time t depends only on S, W and the first t values of each $r[v,\cdot]$. However, $\overline{C}(A,G,S,W)$ need not be finite). Finally, $\overline{C}(A,G) = \sup_{S,W} \overline{C}(A,G,S,W)$.

3.2. Extensions to the model

(1) The ring can be bi-directional — there is an edge (and a buffer) also from v_i to v_{i-1}. Two variants may be considered: one in which all processors agree on their local directionality, and another in which no such agreement is promised.

(2) The topology of the network may be any directed graph not only a ring.

(3) The processors may be nondeterministic. For a given r and S we may ask whether there is a (message or state) terminating computation of the processors and what is the probability of termination.

3.3. Choosing a leader of a ring — a preview

Angluin [A] has shown that there does not exist an algorithm to find a leader of a ring of indistinguishable processors. Her argument considers a ring of four processors where the antipodal processors are always in the same state. When a processor decides that it is the leader, then its image comes to the same conclusion, thus either two leaders are chosen or the algorithm does not terminate.

Lemma 3.1: Let A be a election algorithm in a ring of n processors that know n. If A may terminate with either 0 or more than one leader, then there exists an algorithm A' with the following properties:

(i) A' always selects a unique leader.

(ii) If A terminates correctly then A' also terminates correctly.

(iii) If A terminates with probability 1 and chooses a unique leader with probability $p > 0$ then A' terminates correctly with probability 1.

Proof: Algorithm A might terminate incorrectly either by

(i) Selecting more than one leader, or

(ii) Choosing no leader.

To overcome the first difficulty, whenever a leader is chosen, it sends a verification message along the ring. By having a special field in the message to count the number of processors the message has travelled, the sender can recognize its own messages (compare the value of the field to n). If additional candidates for being a leader are found then the algorithm is reinvoked. To distinguish between the various invocations, phase numbers may be added.

To overcome the second difficulty, the algorithm is reinvoked whenever no leader has been chosen. When n is known, a suitable distributed termination detection algorithm adapted to such situations can be superimposed on any algorithm [BF]. The technique to detect termination distributedly [SF] can be used by every processor individually to check whether all the other processors have halted.

Incorrect termination can be replaced, therefore, by reinvocation of the algorithm. Assume that A terminates with probability 1 and chooses a unique leader with probability $p > 0$. The probability for A to be invoked exactly k times is $p(1-p)^{k-1}$ and thus the average number of invocations is $1/p$ which proves that A' has probability 0 to diverge. □

3.4. The Algorithm

Consider the following *selection algorithm:*

> Initially all processors are *active.* In a general phase, some processors may have become *inactive* whereupon they only relay all the messages received. Each active processor performs the

following step c times or until it becomes inactive.

(1) Choose 0 or 1 each with probability ½, and send the choice to the next active processor.

(2) An active processor becomes inactive if it chose 0 and the active processor preceding it chose 1.

The above simple algorithm uses ideas of Dolev, Klawe and Rodeh [DKR] and Peterson [Pe]. When the algorithm ends, at least one processor remains active since a processor may become inactive only if it chose 0 while another processor chose 1. However, the above algorithm does not guarantee that a leader is chosen since it is possible that more than one processor remained active. We follow Lemma 3.1 and introduce a *verification phase:*

> Every active processor sends a counter around the ring to check whether it is the only active processor.

We first give an intuitive reason why for $c = 5\log n$ the complexity is $O(n\log n)$ bits: The selection phase consists of *c rounds.* The probability that an active processor becomes inactive in a round is ¼ (it must have chosen 0 and its predecessor chose 1). The expected number of processors which become inactive in round 1 is $\sum_{i+1}^{n} Pr$(processor i has become inactive) = $n/4$. And in general, one quarter of the active processors of a round become inactive (except if there is only one active processor). Thus, the expected number of rounds until only one processor remains active is $\log_{4/3} n$. If we choose $c > 2\log_{4/3} n \approx 4.8188\log n$ then the probability that more than one processor remains active after c rounds is small. Thus, the expected bit complexity of repetition is negligible.

The bit complexity of each round of the selection phase is n, since on each edge exactly one bit is transmitted. Thus, the total bit complexity of the selection phase is $c \cdot n = O(n\log n)$. In the verification phase there are a active processors, each sends a counter of $\log n$ bits which travels distance n. The total is $an\log n$. Since the expected value of a is 1, the expected cost of phase 2 is $O(n\log n)$. Since the probability of repetition is small the total cost is also $O(n\log n)$.

We now give a formal analysis due to Prof. M. Hofri: Let X_t be the number of active processors which became inactive in stage t of the algorithm. Further notation:

$$\begin{aligned} D_t &= \sum_{i=1}^{t} X_i \\ N_t &= n - D_t \\ Q_t(z) &= \sum_{d\geq 0} Pr\,(D_t{=}d) z^d . \end{aligned}$$

To develop a formula for the distribution of X_1 notice that in order for k processors to become inactive in the first round, the n choices of the processors must form $2k$ runs of 0 and 1's. These runs can start at $2k$ places; thus there are $2\binom{n}{2k}$ ways for k processors to become inactive (the factor 2 arises since given the border points there are two possibilities: processor v_0 could have chosen 0 or 1). Since the processors can make their choices in 2^n ways

$$Pr\,(X_1{=}k) = 2^{-n+1}\binom{n}{2k}.$$

The generating function of X_1 is immediate

$$Q_1(z) = \sum_{k\geq 0} Pr\,(X_1{=}k) z^k {=} 2^{-n}[(1+\sqrt{z})^n + (1-\sqrt{z})^n],$$

The first and second moments are easy to compute:

$$E\,(X_1) = Q'_1(1) = \frac{n}{4} \qquad (n > 1)$$

$$E\,(X_1^2) = Q''_1(1) + E\,(X_1) = \frac{n\,(n+1)}{16}.$$

While $N_{t-1} > 1$, the successive stages are independently conditioned on the number of active processors at each stage. The results for X_1 may be used for X_t, when n is replaced by $N_{t-1} = n - D_{t-1}$. To overcome the difficulty of $n \leq 1$ we define a surrogate process $\tilde{N}_t$ such that

$$E\,(\tilde{X}_t \mid \tilde{D}_{t-1}{=}d) = \frac{n-d}{4}$$

Thus

$$N_t = \begin{cases} N_t & N_t > 1 \\ 1 & \text{otherwise.} \end{cases}$$

The connection between the processes is that $\tilde{N}_t \leq 1$ implies that $N_t{=}1$. Thus, we can investigate the former instead of the latter.

Lemma 3.2

$$E(\tilde{D}_t) = n(1-(\tfrac{3}{4})^t)$$

$$V(\tilde{D}_t) = \frac{n}{3}[(\tfrac{3}{4})^t-(\tfrac{3}{4})^{2t}].$$

Proof:

By definition,

$$E(\tilde{X}_t \mid \tilde{D}_{t-1}) = \frac{n-\tilde{D}_{t-1}}{4},$$

$$E(\tilde{X}_t) = \sum_i Pr(\tilde{D}_{t-1}=i)\frac{n-i}{4} = \frac{n-E(\tilde{D}_{t-1})}{4},$$

$$E(\tilde{D}_t) = E(\tilde{D}_{t-1}+\tilde{X}_t).$$

Solving the recurrence relation we get

$$E(\tilde{D}_t) = n(1-(\tfrac{3}{4})^t).$$

Similarly,

$$E(\tilde{X}_t^2 \mid \tilde{D}_{t-1}=d) = \frac{(n-d)(n-d+1)}{16}$$

$$E(\tilde{X}_t^2) = \frac{1}{16}\sum_{d\geq 0} Pr(\tilde{D}_{t-1}=d)(n^2-(2n+1)d+d^2+n) = \frac{E(\tilde{D}_{t-1}^2)}{16} - n\frac{n-(2n+1)(\tfrac{3}{4})^{t-1}}{16}.$$

Also

$$E(\tilde{X}_t\tilde{D}_{t-1}) = \sum_{d\geq 0} Pr(\tilde{D}_{t-1}=d)d\frac{n-d}{4} = \frac{n}{4}E(\tilde{D}_{t-1})-\frac{1}{4}E(\tilde{D}_{t-1}^2) = \frac{n^2}{4}\Big(1-(\tfrac{3}{4})^{t-1}\Big) - \frac{1}{4}E(\tilde{D}_{t-1}^2).$$

Since

$$E(\tilde{D}_t^2) = E((\tilde{X}_t+\tilde{D}_{t-1})^2) = E(\tilde{D}_{t-1}^2) + E(\tilde{X}_t^2)+2E(\tilde{X}_t\tilde{D}_{t-1})$$

we get,

$$E(\tilde{D}_t^2) = \frac{9}{16}E(\tilde{D}_{t-1}^2) + n\frac{1-6n}{16}(\tfrac{3}{4})^{t-1} + \frac{7}{16}n^2$$

$$= n^2 + n\frac{1-6n}{4}(\tfrac{3}{4})^{t-1} + n\frac{3n-1}{4}(\tfrac{3}{4})^{2t-1} = n^2+n\frac{1-6n}{3}(\tfrac{3}{4})^t+n\frac{3n-1}{3}(\tfrac{3}{4})^{2t}.$$

$$V(\tilde{D}_t) = E(\tilde{D}_t^2)-E^2(\tilde{D}_t) = \frac{n}{3}[(\tfrac{3}{4})^t-(\tfrac{3}{4})^{2t}].$$

□

Lemma 3.3

$$Pr(N_t > 1) \le \frac{n}{3}(3/4)^t + n\frac{3n-1}{3}(3/4)^{2t}.$$

Proof: By Chebychev's inequality

$$Pr(N_t > 1) \le E(N_t^2) = E((n - \tilde{D}_t)^2) = \frac{n}{3}(3/4)^t + n\frac{3n-1}{3}(3/4)^{2t}. \qquad \square$$

Corollary: For $\frac{c(n)}{\log n} > \frac{2}{\log_2 4/3} \approx 4.8188417$ the expected communication complexity of the algorithm is $O(n\log n)$ bits.

Proof: In this case $(3/4)^t = o(n^{-2})$, thus, $Pr(N_t > 1) = o(n^{-1})$ and the average cost of the selection phase is $O(n\log n)$. The cost of the verification phase is $O(n\log n)$ too. Since the probability that a leader has not been chosen after the selection phase is $o(n^{-1})$ the expected cost of additional phases is negligible. □

4. FINDING THE SIZE OF THE RING

4.1. Definition of the problem

Let R be a ring with n processors, such that every processor v has a special register b_v. An algorithm *A finds the size of R*, if whenever A is applied to R and (state or message) terminates, every b_v contains the value n.

Lemma 4.1: There exists an algorithm to find the size of a ring with a leader, which requires n time units and n bits for a ring of size n. If the ring is asynchronous then finding n requires n messages and $n\log n$ bits. Moreover, these bounds are tight. □

Thus, we concentrate on finding n, in a setup of indistinguishable processors.

4.2. Impossibility Results

An algorithm A is *partially correct* with respect to a predicate Ψ if for every ring, Ψ holds whenever A (state or message) terminates. Thus, A is a partially correct algorithm for finding n if for every ring R, whenever A terminates, $b_v = n$ for all $v \in R$. Note that the trivial algorithm which never terminates is vacuously partially correct.

Theorem 4.1: A processor terminating algorithm which is partially correct for a ring of size N cannot be partially correct for a ring of size $2N$.

Proof: Similar to that of Angluin [A]. □

Corollary: There exists no partially correct processor terminating algorithm to calculate the size of all rings. □

Since an algorithm cannot be always correct, we investigate algorithms which are correct most of the time. An algorithm A is *correct with probability* p with respect to Ψ if for every ring, wakeup messages distribution W and schedule S, A terminates and the probability that Ψ holds is greater than p. The following theorem shows that if we insist on processor termination the probability of error cannot be bounded away from 1.

Theorem 4.2: There does not exist a processor terminating asynchronous algorithm to calculate the size of the ring which is correct with probability $\alpha > 0$.

Proof: Suppose to the contrary that such an algorithm A existed. I.e., there exists a ring $R = (v_0, \ldots, v_{n-1})$, such that for every schedule S and wakeup messages W, with probability at least α A finds the size of R to be n. In particular, we may assume that the algorithm starts with the single wakeup message $W = \{(v_0, v_1)\}$, and that the schedule "sweeps around the ring", i.e., $S = ((v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_0), (v_0, v_1), \cdots)$. (At time slot j, S enables the edge $(v_{j \bmod n}, v_{j+1 \bmod n})$.)

We intend to show that with probability larger than $1-\alpha$ certain processors in a ring R', much larger than R, run into the erroneous conclusion that the size of the ring is also n. To this end, we let several processors simulate each processor of R. The difficulty with the simulation is that the behavior of each processor depends not only on the messages it receives but also on its random sequence. Since, there is probability zero that two processors have exactly the same random sequences, if the state incorporates the prefix of a random sequence, then there is probability zero that the states are identical. Therefore, we first exhibit a series of messages which have positive probability of occurring and which cause the processors to decide that the size of the ring is n. Then we show that the probability of random sequences causing a processor to participate in such a computation must be positive. Therefore, the probability that other processors have similar sequences is $p > 0$. By making the ring larger, the probability tends to 1 that some other processors exhibit the same behavior as the processors of the ring of size n.

An *execution* $\mathbf{x}$ is a triple $<\mathbf{m},\mathbf{e},b>$ where $\mathbf{m}=(m_1, \ldots, m_t)$ is a sequence of t messages, $\mathbf{e}=(e_1, \ldots, e_t)$ a prefix of the schedule S and b an integer. The meaning is that at time slot j, e_j was enabled and the message m_j was read. If v_i is the processor which read m_t then b is the value written into b_{v_t} and v_i is in a halting state. An execution is *correct* if $b=n$, the size of the ring. (Note that in a correct execution all processors v_k set b_{v_k} to n, thus if $v_i = n$, so is the value of all the other b_{v_k}'s.) Since A processor terminates correctly on R there exist such finite executions. For an execution $\mathbf{x}$, let $B(\mathbf{x})$ be the bit complexity of $\mathbf{x}$ if $\mathbf{x}$ is correct and zero otherwise. Since $\mathbf{x}$ is a random variable, so is B. Moreover,

Lemma 4.2: For A, R, S and W as above, there exists a positive probability $\beta > 0$ and a natural number k such that

$$Pr(B = k) = \beta.$$

Proof: If the lemma does not hold then for all k, $Pr(B = k) = 0$. Thus,

$$\sum_{k=1}^{\infty} Pr(B = k) = 0,$$

which means that with probability 1, B is infinite. However, since in each step only a finite number of bits is transmitted, B can be infinite only when the number of steps is also infinite, i.e., when the algorithm diverges. Thus if the lemma does not hold, then with probability 1 the algorithm does not halt correctly. In particular, the probability that the algorithm correctly computes the size of the ring is smaller than α. □

Remark: A partially correct algorithm may be applied to rings of various sizes. Therefore, there must be some communication, i.e., $Pr(B=0) = 0$.

Lemma 4.3: There exists a correct execution $\mathbf{x}$ and $\gamma > 0$ such that with probability γ (for the above S and W) the execution $\mathbf{x}$ is observed.

Proof: By Lemma 4.2 and the subsequent remark there exist $\beta > 0$ and $k > 0$ such that with probability β the bit complexity is k. There is only a finite number of correct executions with bit complexity k, thus there exists a correct execution $\mathbf{x}$ and $\gamma > 0$ such that there is probability γ that $\mathbf{x}$ occurred. □

Let R' be a ring consisting of N *segments* each of which is a copy of $v_0, \ldots, v_{n-1}$, i.e.,

$$R' = (u_{0,0}, u_{0,1}, \ldots, u_{0,n-1}, u_{1,0}, \ldots, u_{N-1,n-1}).$$

W' consists of a wakeup message for the first edge in each segment, namely,

$$W' = \{(u_{N-1,n-1}, u_{0,0}), (u_{0,n-1}, u_{1,0}), \ldots, (u_{N-2,n-1}, u_{N-1,0})\}.$$

To facilitate the discussion we partition the time slots into *superslots* each consisting of N time slots. (I.e., the first superslot consists of time slots $1, \ldots, N$, the second superslot of time slots $N+1, \ldots, 2N$ etc.) Finally, S' sweeps around the ring segment by segment. If in the j-th time slot the edge (v_k, v_{k+1}) was enabled, then in the j-th superslot all the N edges of the form $(u_{i,k}, u_{i,k+1})$ are enabled. Namely,

$$S' = ((u_{0,0}, u_{0,1}), (u_{1,0}, u_{1,1}), \ldots, (u_{N-1,0}, u_{N-1,1}), (u_{0,1}, u_{0,2}), \ldots, (u_{N-1,1}, u_{N-1,2}), \cdots), \cdots).$$

Consider the execution **x** of Lemma 4.3, let $X_j^\tau(w)$ denote the event that during the first τ superslots $w \in R$ behaved like $v_j \in R$ did in the first τ time slots of **x**, (i.e., w sent the same messages as v_j and set b_w if and only if b_{v_i} was set).

Lemma 4.4:

(i) $Pr(X_j^0(w)) = 1$ $\qquad w \in R'$;

(ii) $Pr(X_j^\tau(u_{i,j}) \mid X_{j-1}^{\tau-1}(u_{i,j-1})) \geq \gamma$ $\qquad (0 \leq i \leq N-1,\ 1 \leq j \leq n-1,\ 1 \leq \tau \leq t)$;

(iii) $Pr(X_0^\tau(u_{i,0}) \mid X_{n-1}^{\tau-1}(u_{i-1,n-1})) \geq \gamma$ $\qquad (1 \leq i \leq N-1,\ 1 \leq \tau \leq t)$.

Proof:

(i) When $\tau = 0$ neither R nor R' issued any communication, thus all processors in both rings are in their initial state, and thus have exhibited identical behavior.

(ii) If during the entire algorithm $u_{i,j}$ received the same messages as v_j then there is probability $p_j \geq \gamma$ that it behaved like v_j. In particular, after receiving the messages of the first τ time slots the probability for the same behavior up to that point is $\geq \gamma$.

(iii) Similar to (ii). □

During the first t superslots a vertex $w \in R'$ is not effected by vertices at distance $> t$. Therefore, we partition R' into disjoint *supersegments* each consisting of t segments.

Lemma 4.5: Let w_τ be the τ-th vertex of a supersegment. Then $Pr(X_j^\tau(w_\tau)) \ge \gamma^\tau$, where $j = \mathrm{mod}(\tau,n)$.

Proof: By induction on τ.

Basis: $\tau = 0$. Follows from Lemma 4.4 (i).

Induction step: $\tau \ge 1$. If $j \ge 1$, then by Lemma 4.4 (ii) $Pr(X_j^\tau(w_\tau) \mid X_{j-1}^{\tau-1}(w_{\tau-1})) \ge \gamma$. The induction hypothesis states that $Pr(X_{j-1}^{\tau-1}(w_{\tau-1})) \ge \gamma^{\tau-1}$. Therefore, by the law of conditional probabilities,

$$Pr(X_j^\tau(w_\tau)) = Pr(X_j^\tau(w_\tau) \mid X_{j-1}^{\tau-1}(w_{\tau-1}))Pr(X_{j-1}^{\tau-1}(w_{\tau-1})) \ge \gamma \cdot \gamma^{\tau-1} = \gamma^\tau.$$

The case $j = 0$ follows similarly from Lemma 4.4 (iii). □

Conclusion of the proof of Theorem 4.2: We have shown that with probability at least $p = \gamma^t$ the t-th vertex of a supersegment concludes erroneously that the size of R' is n.

Moreover, the behavior of the t-th vertex of disjoint supersegments is statistically independent. Thus, the probability that no vertex of R' concludes that the size is n is at least $\left(1-\gamma^t\right)^{\lfloor N/t \rfloor}$. The latter expression tends to zero when N approaches infinity. □

4.3. Finding n and choosing a leader when $N \le n < 2N$

In this section we give partially correct algorithms to choose a leader when n is known to lie in the interval $[N, 2N-1]$. Since n is not known precisely and the processors do not have unique id's, processors cannot recognize their own messages. However, we may use the fact that a message travelling distance $2N-1$ visits all the processors, and passes through its originator exactly once.

4.3.1. The synchronous model

The proposed algorithm, like the algorithms of Section 3, consists of two phases: selection and verification.

The selection phase is iterated several times until one or more candidates for leadership are chosen. Each iteration lasts $2N-1$ time slots. At the first time slot of each iteration each processor v chooses an id $\in \{0,1\}$, such that with probability c/N $id_v = 1$. A processor with $id = 1$ is called a *candidate.* Every candidate sends a bit in the first time slot and will not pass any additional bits that may be received later. The other processors wait until receiving a bit and only then send it. If no processor chose $id = 1$, then there is no communication and at the end of the iteration all processors have become aware of this, and the selection phase is repeated. The selection phase terminates with an iteration in which there exist one or more candidates.

The verification phase consists of two subphases. During the first subphase some (but not necessarily all) processors know whether a single candidate has been chosen. The remaining processors learn about it in the second subphase.

The first subphase of the verification phase is very similar to the selection phase: The candidates of the selection phase initiate communication (i.e., send a single bit in the first time slot), but do not pass any bits they may have received later. The other processors send a bit only after receiving one. The entire subphase lasts only $N-1$ time slots. If there is only one candidate only one message is sent during this subphase. Since the subphase consists of $N-1<n$ time slots that message does not make a complete tour thus the candidate does not receive any message during this subphase. If, however, there is more than one candidate, since $n \le 2N-1$ there exist two candidates at distance $\le N-1$ from each other. One of those candidates receives a message (originating from the other one) thus if a candidate receives a message in that subphase it knows that there are more than one candidate. Consequently, at the end of the first subphase, if there is more than one candidate then there exists a candidate that knows this (not necessarily all of them know). However, if there is a single candidate, then at the end of this subphase it cannot be sure that it is the only one. The purpose of the second subphase is to notify all the processors.

The second verification subphase is similar to the previous phases, it lasts $2N-1$ time slots. The candidates that know that there are more than one candidate initiated communication (and do not pass any bits that may have arrived later). The remaining processors just pass any bits they receive. Consequently, no bits were sent during the subphase if and only if there remains a single candidate. By sensing the absence of communication during the subphase, all processors can discover that there is a single candidate and it becomes the leader. If any bit is sent during the second subphase, every processor receives a bit and there must be more than one candidate. In this case the entire algorithm is reinvoked.

To analyze the complexity, first note that if no candidate was chosen in the selection phase then there was no communication. Thus, regardless of the number of times the selection phase was repeated, in each invocation of the algorithm the selection phase requires exactly n bits of communication. The first verification subphase requires $N-1$ bits if a single candidate remained and at most n bits otherwise. The second subphase always requires exactly n bits. Thus, if a single candidates remains the invocation requires $n+N-1$ bits, otherwise the invocation requires $\leq 3n$ bits. The time of an invocation is $3N-2+k_{SEL}(2N-1)$, where k_{SEL} is the number of times the selection algorithm is iterated within a single invocation of the algorithm. The expected value of k_{SEL} and the probability that the algorithm is reinvoked depend on c, N and n.

Let the random variable $M=M(c,n,N)$ denote the number of candidates.

$$Pr(M=i) = \binom{n}{i}\left(c/N\right)^{i}\left(1-\frac{c}{n}\right)^{n-i}.$$

(Note that since c/N is the probability that a processor chose $id=1$, $c < N$.) The probability that a single iteration of the selection phase is sufficient

$$Pr_{SEL} = Pr(M>0) = 1 - Pr(M=0) = 1 - \left(1-\frac{c}{N}\right)^{n} \geq 1 - (1-\frac{c}{N})^{N} > 1 - e^{-c}.$$

The expected number of iterations of the selection phase:

$$\bar{k}_{SEL} = {Pr_{SEL}}^{-1} < \frac{1}{1-e^{-c}}.$$

The probability that the entire algorithm is performed once

$$Pr_{ALG} = Pr(M=1 \mid M>0) = \frac{Pr(M=1 \;\&\; M>0)}{Pr(M>0)} = \frac{Pr(M=1)}{Pr_{SEL}} = \frac{\binom{n}{1}\left(\frac{c}{N}\right)^{1}\left(1-\frac{c}{n}\right)^{n-1}}{1-\left(1-\frac{c}{N}\right)^{n}}$$

$$= \frac{n\frac{c}{N}\left(1-\frac{c}{N}\right)^{n-1}}{1-(1-\frac{c}{N})^{n}} = \frac{n\frac{c}{N}}{\left(1-(1-\frac{c}{N})^{n}\right)(1-\frac{c}{N})^{1-n}} = \frac{n\frac{c}{N}}{\left((1-\frac{c}{N})^{-n}-1\right)(1-\frac{c}{N})}.$$

The expected number of iterations of the entire algorithm is

$$\bar{k}_{ALG} = \frac{\left((1-\frac{c}{N})^{-n}-1\right)(1-\frac{c}{N})}{n\frac{c}{N}} \tag{4.1}$$

$$< \frac{(1-\frac{c}{N})^{-n}-1}{c} \leq \frac{(1-\frac{c}{N})^{-(2N-1)}-1}{c} < \frac{\left((1-\frac{c}{N})^{-N}\right)^{2}-1}{c}.$$

Since for $c < N$,

$$(1-\frac{c}{N})^{-N} = \exp\left(-N\ln(1-\frac{c}{N})\right) = \exp\left(N\sum_{k=1}^{\infty}\frac{1}{k}(\frac{c}{N})^k\right) = \exp\left(c+N\sum_{k=2}^{\infty}\frac{1}{k}(\frac{c}{N})^k\right) = e^c\exp\left(N\sum_{k=2}^{\infty}\frac{1}{k}(\frac{c}{N})^k\right)$$

$$< e^c\exp\left(\frac{N}{2}\sum_{k=2}^{\infty}(\frac{c}{N})^k\right) \le e^c\exp\left(\frac{N}{2}(\frac{c}{N})^2(1-\frac{c}{N})^{-1}\right) \le e^c\exp\left(\frac{c^2}{2(N-c)}\right),$$

we get

$$\overline{k}_{ALG} < \frac{1}{c}\left[\left(e^c\exp\left(\frac{c^2}{2(N-c)}\right)\right)^2 - 1\right] = \frac{1}{c}\left[\left(e^{2c}\exp\left(\frac{c^2}{N-c}\right)\right) - 1\right]$$

Theorem 4.3: In a synchronous ring a leader can be chosen within expected bit complexity $\overline{C} = (3\overline{k}_{ALG}-2)n+N-1$ and expected time $\overline{T} = \overline{k}_{ALG}\cdot(3N-2+\overline{k}_{SEL}(2N-1))$, where $\overline{k}_{SEL} < \dfrac{1}{1-e^{-c}}$ and

$$\overline{k}_{ALG} < \frac{1}{c}\left[\left(e^{2c}\exp\left(\frac{c^2}{N-c}\right)\right) - 1\right] .$$

Remark: There exists a tradeoff between $\overline{C}$ and $\overline{T}$: Consider equation (4.1), if n and N are fixed and c decreases to 0 then $\overline{k}_{ALG}$ converges to 2, thus $\overline{C}$ converges to $n+N-1$. However, this is infeasible since $\overline{k}_{SEL}$, and with it $\overline{T}$, approach ∞.

Corollary: In a synchronous ring a leader can be chosen within expected bit complexity $\overline{C} = 10n+N-1$ and expected time $\overline{T} = 35N$.

Proof: Choose $c = ½$. The entire problem is trivial for N=1. Thus we need consider only N≥2, since the bound for $\overline{k}_{ALG}$ is a decreasing function of N,

$$\overline{k}_{ALG} < 2\left(e\cdot\exp(\frac{¼}{N-¼})-1\right) < 2\left(e\cdot\exp(\frac{¼}{2-¼})-1\right) \approx 4.2714295.$$

Thus the expected bit complexity is

$$\overline{C} = 3n\,(\overline{k}_{ALG}-1)+n+N-1 < 9.8142886n+N-1.$$

Since,

$$\overline{k}_{SEL} < \frac{1}{1-e^{-½}} \approx 2.5415$$

the expected time is

$\overline{T} = \overline{k}_{ALG} \cdot (3N - 2 + \overline{k}_{SEL}(2N-1)) < 34.525914N - 19.39872.$ □

After a leader has been elected the size of the ring can be found by sending a bit around the ring and measuring the time slots it takes to travel. To convey the size to all the processors, let the bit traverse the ring twice – the size of the ring is equal to the number of time slots between the first and the second traversal.

4.3.2. The asynchronous model

The algorithm of Section 3.4 is applicable here also. The only difference is that since n is not known the verification message should travel distance $2N-1$. The processor where the message stops knows whether there is more than one leader (the message passes through a leader at least three times). This processor notifies the others whether a unique leader has been elected.

The expected communication complexity is $O(n)$ message which total to $O(n\log N + (2N-1)\log N)$ = $O(n\log n)$ bits.

Finding n requires an additional $O(n\log n)$ bits.

A crossing sequence argument may be used to show that $\Omega(n\log n)$ is a lower bound on deterministic algorithms. (A formal proof may be given using the results of [MZ]).

4.4. A message terminating algorithm to find n with low probability of error

In this section we present an asynchronous algorithm with the following properties:

(i) It always processor terminates. Its time and communication complexities are polynomial.

(ii) The probability of error depends on an external parameter r and can be made arbitrarily small independently of the size of the ring.

During the execution of the algorithm, each processor v has a candidate k_v for the size of the ring; initially k_v=2. A processor may create, pass or cancel messages. The messages are used in tests which may either increase the confidence that $k_v = n$ or show that $k_v < n$. For each value of k_v, rk_v successive tests are conducted. If any of the tests fail or any other indication implies that $k_v < n$ then the processor increases k_v and repeats the test rk_v times (for the new value of k_v). The algorithm terminates when all the processors which tested the value of k_v finished all their tests successfully. At this point no further communication is issued by any of the processors. It will be shown that the algorithm always terminates, on termination all processors have the same value of k_v and that with high probability this value is equal to n.

4.4.1. Description of the algorithm

Prior to termination, some of the processors are active. Each active processor v tries to send a message carrying k_v in the direction of the ring, to a distance k_v. If $k_v = n$ then the message returns to v. If $k_v < n$ then the message terminates at some node $u \neq v$. If $k_u \leq k_v$ and u knew that the message is not its own then u could deduce that $k_v < n$ and thus update k_u to k_v+1, thereby becoming active. The main difficulty is that if n is not known, no processor can identify its own messages with certainty. To help with the identification, each processor v randomly chooses 0 or 1 as its identity – id_v. This identity is incorporated in the message sent by v. Thus, a message m contains three fields $(k_m, id_m, count_m)$. A message starting at v is initially $(k_v, id_v, 1)$.

Assume that a message $(k_m, id_m, count_m)$ terminates at v (thus, $count_m = k_m$). Several cases may arise:

(1) $k_m < k_v$. In this case the message is cancelled.

(2) $k_m > k_v$. The processor sets $k_v := k_m + 1$ and originates the message $(k_v, id_v, 1)$.

(3) $k_m = k_v$ and the processor has not sent a message or $id_m \neq id_v$. The processor v is not the originator of the message, and thus proceed as in (2).

(4) $k_m = k_v$ and $id_m = id_v$ and v has originated a message $(k_m, id_m, 1)$. The processor runs to the (erroneous) conclusion that its own message has returned. If this test has succeeded less than rk_v times, an additional test is initiated.

Here is a more formal description. Initially, each processor v randomly chooses an identity $id_v \in \{0,1\}$ (such that $Prob\,(id_v{=}0) = ½$), and executes the procedure $originate_v(2)$.

```
**procedure** $originate_v(k{:}integer)$
**begin**  $k_v := k$ ;
    $times_v := 1$;
    delete all messages from the input buffer;
    send $(k_v, id_v, 1)$
**end**
```

At any time after initialization, each processor is ready to send and receive messages. If the input buffer of v is not empty, v reads the first message from its input buffer and sends it on.

Upon receiving a message $(k_m, id_m, count_m)$, processor v executes *Program no.* 2. The program uses the procedure $confirm_v$.

4.4.2. Properties of the algorithm

Lemma 4.6: Throughout the algorithm

(i) $count_m \le k_m$

(ii) $k_v \le n$.

Proof:

(i) $count_m$ is increased only at lines L1 and L2 which in turn are called only when $k_m > count_m$.

(ii) $\max_v \{ k_v \}$ is initially $2 \le n$, and it may be increased only at line L3, in which case $k_v = k_m = count_m$ but the message did not originate at v since $id_m \ne id_v$. Thus, there is another node at distance k_v from v. Since $k_v \le n$ we deduce that $k_v < n$ and by increasing k_v by one (ii) still holds. □

The value k_v is nondecreasing. Moreover, each processor originates at most r messages with the same value of k_v. Thus, the total number of messages originating at each processor is rn^2. Consequently,

```
procedure $confirm_v$ ;
begin  $times_v := times_v + 1$;
        send $(k_v, id_v, 1)$
end

begin if $k_m > k_v$ then
        if $k_m > count_m$ then
        begin $originate_v(k_m)$;
L1:             send $(k_m, id_m, count_m + 1)$
        end
        else  $originate_v(k_m + 1)$
         else if $k_m = k_v$ then
        if $k_m > count_m$ then
L2:              send $(k_m, id_m, count_m + 1)$
        else if $id_m = id_v$ then
                 if $r < times_v$ then skip
                 else begin
                                 $id_v :=$ random$\{0,1\}$;
                                 $confirm_v$
                 end
L3: else $originate_v(k_v + 1, 1,$ true$)$
end
```

Program no. 2

the algorithm is finite and we have the following:

Lemma 4.7: The communication complexity is O(rn^3) messages each of O(log$n$) bits. □

Let f_v denote the final value of k_v. The following lemma shows that the algorithm is consistent.

Lemma 4.8: For all processors u,w, $f_u = f_w$.

Proof: Suppose to the contrary, that there exist processors u and w for which $f_w < f_u$. The procedure $originate_u$ increases k_u to f_u. Thus, a message carrying f_u existed. This message could not be cancelled as a result of the arrival of another message. The message could disappear at a node v only if v has sent a message with the same value of f_u exactly r times. Thus, some message carrying f_u succeeds passing through every node, in particular through w, increasing k_w to f_u, thus $k_w \geq f_u$ □

Denote the processors by $v_0, \ldots, v_{n-1}$ and let f denote the common value of f_{v_i}. If $f < n$, then v_0 sends the r messages $(f, id^1(v_0), 1), \ldots, (f, id^r(v_0), 1)$. These messages terminate at v_f, whereupon $k_{v_f}=f$, and v_f also sent identical messages (otherwise v_f would increase k_{v_f} to f+1). Consequently, $id^1(v_0)=id^1(v_f)$ $, \ldots ,$ $id^r(v_0)=id^r(v_f)$.

Let F be the equivalence relation defined as follows: $v_i F v_j$ if there exists an l such that $(j=i+lf \bmod n)$. From the above discussion it is clear that if $v_i F v_j$ then $id^1(v_i)=id^1(v_j)$, $\cdots$, $id^r(v_i)=id^r(v_j)$.

Let $C_0, \ldots, C_{a-1}$ be the equivalence classes of F, each containing $b=n/a$ processors. Let $g = \gcd(f,n)$ then

$$C_i=\{v_j : j=i+lf \bmod n\} = \{v_j : j+i+lg, l=0, \ldots, b-1\}.$$

Therefore, there are $a=g$ equivalence classes. There is probability $2^{-(b-1)}$ that $id^s(v)=id^s(u)$ for all u,v of the same equivalence class, and probability $2^{-(b-1)r}$ that this holds for $s=1, \ldots, r$. If $f < n$ then this must hold for all equivalence classes, thus the probability of error is $2^{-(b-1)ra} = 2^{-(n-a)r}$.

Lemma 4.9:

(i) For n prime the probability of error is $2^{-(n-1)r}$.

(ii) For any n the probability of error is less than or equal to $2^{-nr/2}$.

Proof:

(i) Follows from the fact that if n is prime then $a = \gcd(f, n) = 1$.

(ii) Follows since $a = \gcd(f, n) \le n/2$.

4.4.3. Limiting the size of the buffers

At any time, there are at most n messages. This follows since initially every processor sent at most one message, and new messages are generated only when old ones are cancelled.

This implies that the size of the input buffers does not exceed n. If n is large, the space requirement, being $O(n\log n)$ becomes excessive. However, the size of each buffer can be limited to two, provided a processor can receive a message only when its output buffer is not full. Deadlock does not occur since not all buffers can be full. The space requirement is, thereby, reduced to $O(\log n)$ bits per processor.

4.4.4 Summary

Substituting $\varepsilon = 2^{-nr/2}$ yields,

Theorem 4.4 For all $\varepsilon > 0$ there exists a message terminating algorithm to find n on an asynchronous ring, with error probability ε, communication complexity $O(n^2 \log \varepsilon^{-1})$ messages of $O(\log n)$ bits and whose space requirements is $O(\log n)$ bits per processor.

5. CONCLUSIONS

Since its conception, distributed programming had to face the problem of symmetry (for example, two processors demanding the same resource). Here we have investigated the effects of symmetry on distributed and randomized algorithms. Because the processors cannot recognize their own messages, some problems which are solvable in an asymmetric network have no algorithmic solution for the symmetric case. Other problems are solvable by algorithms of greater complexity because each processor acts independently and the same message is sent many times. A natural continuation is to look at other types of networks. Perhaps, asymmetric networks of identical processors can take advantage of the asymmetry, even if the topology of the network is not known.

It should be pointed out that our negative results are applicable to all network-independent algorithms. I.e., there exists no terminating algorithm to choose a leader or to find *n*.

Other interesting problems are:

(i) Reliably maintaining a leader in a symmetric network subject to processor and communication failure.

(ii) Developing lower bounds on the complexity of probabilistic algorithms for symmetry breaking both for rings and for general networks.

REFERENCES

[A] D. Angluin, *Local and global properties in networks of processes,* 12th Annual ACM Symp. on Theory of Computing, Los Angeles, California, 82-93, (April 1980).

[AAHK] K. Abrahamson, A. Adler, L. Highman and D. Kirkpatrick, *Probabilistic solitude verification on a ring,* 5th ACM Symp. on Principles of Distributed Computing (PODC), 161-173 (August 1986).

[BF] L. Bouge and N. Francez, *A compositional approach to superimposition,* The 15-th ACM Symp. on the Principles of Programming Languages (POPL), San-Diego, CA, 240-249, (Jan. 1988).

[Bo] L. Bouge, *On the existence of a symmetric algorithm to find leaders in networks of communicating sequential processes*, TR 86-18, LITP, Univ. Paris 7, (1986), to appear in Acta Informatica.

[Bu] J.E. Burns, *A formal model for message passing system,* TR-91, Indiana University, (Sept. 1980).

[CR] E. Chang and R. Roberts, *An improved algorithm for decentralized extrema-finding in circular configurations of processes,* Comm. ACM, vol. 22, 281-283, (1979).

[DG] P. Duris and Z. Galil, Two lower bound in asynchronous distributed computation, 28 FOCS 326-330, (1987).

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

[F1] R. Franklin, *On an improved algorithm for decentralized extrema-finding in circular configuration of processes,* Commum. ACM, vol 25, 336-337, (1982).

[F2] N. Francez, *Fairness,* Text and Monographs in Computer Science Series, D. Gries (ed.), Springer-Verlag, (1986).

[HS] D.S. Hirschberg and J.B. Sinclair, *Decentralized Extrema-Finding in Circular Configurations of Processes,* Comm. ACM vol. 23, (Nov. 1980).

[IMN] A. Itai, S. Moran and N. Navoni - in preparation.

[IR] A. Itai and M. Rodeh, *Symmetry breaking in distributed network,* Proceeding of the 22nd Annual IEEE Symp. of Foundations of Computer Science (FOCS), 245-260, (1981).

[L] G. LeLann, *Distributed systems - toward a formal approach,* Information Processing 77, North-Holland Pub. Co., Amsterdam, 155-160.

[Lem] A. Lempel, private communication.

[MM] G.A. Milne and R. Milner, *Concurrent processes and their syntax,* Journal ACM vol. 26, 302-321, (1979).

[MZ] Y. Mansour and S. Zaks, *On the bit complexity of distributed computations in a ring with a leader,* Inf. and Computation, vol. 75, 162-177, (1987).

[Pa] J. Pachl *A lower bound for probabilistic distributed algorithms*, Research report CS-85-25, Univ. of Waterloo, (August 1985).

[Pe] G.L. Peterson, *An* $O(n \log n)$ *unidirectional algorithm for the circular extrema problem*, IEEE Transactions on Programming Languages and Systems, vol. 4, 758-762, (1982).

[R1] M.O. Rabin, *Probabilistic algorithms*, Proc. Symp. on New Directions and Recent Results in Algorithms and Complexity, J.F.Traub ed., Academic Press, (1976).

[R2] M.O. Rabin, *The choice coordination problem*, Acta Informatica 17, 121-134, (1982).

[SF] N. Shavit and N. Francez, *A new approach to detection of locally indicative stability, Proc. 13 ICALP, Rennes, France, July 1986, LNCS 226 (L. Kott, ed.) Springer-Verlag, 1986.*