Randomized Function Evaluation on a Ring**

by
Karl Abrahamson*
Andrew Adler†
Lisa Higham*
David Kirkpatrick*

Technical Report 87-20
May 1987

*Department of Computer Science
†Department of Mathematics

** This research was supported in part by the Natural Sciences and Engineering Research Council of Canada and the Killam Foundation.

Abstract

Let R be a unidirectional asynchronous ring of n processors each with a single input bit. Let f be any cyclic non-constant function of n boolean variables. Moran and Warmuth [8] prove that any *deterministic* algorithm for R that evaluates f has communication complexity $\Omega(n \log n)$ bits. They also construct a cyclic non-constant boolean function that can be evaluated in $O(n \log n)$ bits by a deterministic algorithm.

This contrasts with the following new results:

1. There exists a cyclic non-constant boolean function which can be evaluated with expected complexity $O(n\sqrt{\log n})$ bits by a *randomized* algorithm for R.
2. Any *nondeterministic* algorithm for R which evaluates any cyclic non-constant function has communication complexity $\Omega(n\sqrt{\log n})$ bits.

1 Introduction

An asynchronous unidirectional ring is one of the simplest network topologies. Because it exhibits some phenomena that could be considered typical of more elaborate networks, it provides a good starting point for the study of distributed computation. Hence many fundamental problems on rings have been widely studied under various additional assumptions about the ring.

Function evaluation is one typical problem. A ring of n processors, π_1, $\ldots$, π_n, each with a single input value (say $i_1, \cdots, i_n$ respectively), must cooperate to compute $f(i_1, \cdots, i_n)$ for some n-variable function f. The general question of the minimum communication complexity required to compute any non-trivial function on a ring was first addressed by Moran and Warmuth in [8]. They focus on the complexity of non-constant functions because any constant function can be evaluated without any communication. Since computation is on a ring of identical processors, the function f is assumed to be *cyclic*, that is $f(i_1, \cdots, i_n) = f(i_j, \cdots, i_n, i_1, \cdots, i_{j-1})$ for any $1 \leq j \leq n$. They prove that:

1. If g is any non-constant cyclic function of n variables, then the complexity of evaluating g on a ring of n identical *deterministic* processors is $\Omega(n \log n)$ bits of communication and
2. There is a is non-constant cyclic boolean function f, such that f can be evaluated in $O(n \log n)$ bits of communication on a ring of n *deterministic* processors.

It has become increasingly apparent that randomization can be a powerful tool in distributed and parallel computing, particularly when used as an aid to breaking symmetry. The question therefore arises whether the bounds described above remain true if the processors are not deterministic but rather have access to independent random number generators. This paper answers this question. We exhibit a non-constant cyclic boolean function f and a randomized distributed algorithm which evaluates f on a ring of n processors using expected complexity $O(n\sqrt{\log n})$ bits for any input. We also prove that $\Omega(n\sqrt{\log n})$ bits are required to evaluate any non-constant cyclic function on a ring.

The lower bound actually holds for a more powerful model of computation than that used to achieve the upper bound. This model is described in the next section. Section 3 provides the proof of the lower bound. Section 4 presents a function with minimum complexity, and section 5 reviews some remaining open problems.

2 Model

Our objective is to study the inherent bit complexity of randomized distributed algorithms that evaluate functions on asynchronous rings. A distributed algorithm can be viewed as an assignment of processes to processors. So it suffices to model computations induced by cyclic sequences of processes. In order to describe a computation, we first define a process and state some useful properties of rings of processes. The relationship between distributed algorithms and rings of processes is then made precise on order to highlight the generality of the lower bounds that follow.

2.1 Processes

The following description of a process incorporates two non-restrictive assumptions [10], namely, that messages are self-delimiting, and that communication is message driven with at most one message sent in response to receipt of a message.

A *message* is an element of $M = \{0,1\}^* \cdot \square$. The symbol $\square$ is called the end-of-message marker. A *history* is a sequence of messages. If h is a history, then $|h|$ denotes the number of messages in h, and $\|h\|$ denotes the length of the binary encoding of h using some fixed encoding scheme to encode each symbol in $\{0, 1, \square\}$. Note that any history has a unique parse into a sequence of messages.

A *process* π is modelled as a pair of mappings which describe the next

output message (possibly null) and the next state of π as a function of its current state and current input message. A process' state encodes its entire history so far, and consequently the state set is not bounded. Process π is an *initiator* if it produces an output message from its initial state, (that is, before the arrival of the first message). Otherwise it is a *non-initiator*.

Let $\pi_1, \ldots, \pi_n$ be a sequence of processes. We use $\pi_{1,n}$ to abbreviate $\pi_1, \ldots, \pi_n$. There is a sequence of histories, $C = h_1, \cdots, h_n$, called a *computation* associated with $\pi_{1,n}$ in the natural way. Each history h_i is composed of a sequence of messages $m_{i_1} \cdots m_{i_{r_i}}$. If π_i is an initiator, then m_{i_1} is the message produced by π_i from its initial state. The computation is then determined inductively by applying the mappings defined by π_1 through π_n and letting successive output messages of π_i be successive input messages of π_{i+1}. (Indices are reduced in the obvious way.) The *complexity* of a computation $C = h_1, \cdots, h_n$ is $\sum_{i=1}^{n} \|h_i\|$.

We distinguish a subset $M_a \subseteq M$ called *accepting* messages, and a subset $M_r \subseteq M$ called *rejecting* messages. A history is an *accepting* history (respectively, *rejecting* history) if and only if its *last* message is an accepting message (respectively, rejecting message). An accepting or rejecting history is a *decisive* history. A computation $h_1, \cdots, h_n$ of $\pi_1, \ldots, \pi_n$ is *accepting* if *every* h_i is an accepting history, and is *rejecting* if *every* h_i is a rejecting history.

We are interested in *distributive termination* for function evaluation, that is, processes must reach irreversible conclusions. This is modelled by insisting that processes never output another message after sending a decisive message. Non-distributive termination permits a process to reach a tentative decision which it may revoke upon receipt of another message. This weaker form of termination is of little interest in the context of function evaluation, since this termination cannot be detected. In any case the reader will readily construct a non-distributively terminating deterministic algorithm for AND which requires only $O(n)$ bits.

A number of properties and lemmas follow immediately from these definitions. They allow us to manipulate computations, building new ones from old ones.

Lemma 2.1: If $C = h_1, \cdots, h_k$ is a computation with complexity less than b, and all h_i are distinct, then $k < \frac{3b}{\log b}$.

Proof: At least $(k \log k)/2$ bits are required to encode k distinct strings. Hence $(k \log k)/2 < b$ implying $k < \frac{3b}{\log b}$. ∎

Property 2.2: If $C = h_1, \cdots, h_n$ is a computation of $\pi_1, \ldots, \pi_n$ with $h_i = h_j$ for some $i < j$, and π_1 is the only initiator, then $C' = h_1, \cdots, h_i, h_{j+1}, \cdots, h_n$ is a computation of $\pi_1, \cdots, \pi_i, \pi_{j+1}, \cdots, \pi_n$.

Property 2.3: If $C = h_1, \cdots, h_n$ is a computation of $\pi_{1,n} = \pi_1, \ldots, \pi_n$ then $C^r = (h_1, \cdots, h_n)^r$ is a computation of $(\pi_{1,n})^r = (\pi_1, \ldots, \pi_n)^r$ where $(x)^r$ has the usual interpretation: the concatenation of r copies of sequence x.

2.2 Algorithms

The processes just described are deterministic. In order to explore function evaluation by randomized algorithms, we need to incorporate input values and random choices into the model. Both of these features are achieved simultaneously by relaxing the way in which processes are assigned to processors, rather than by generalizing the notion of a process.

In the natural description of a randomized distributed algorithm, a process' next state and output message are determined by its current state, its last input message and the result of a random experiment. Random choices occur throughout the run of the algorithm. But these random choices can be simulated as a single random choice by each processor at the beginning of the algorithm. A processor randomly chooses a function from internal state, input message pairs to internal state, output message pairs. (Essentially, the processor pre-selects all its random coin tosses.)

Inputs are incorporated into initial states of processes. If a processor has a given input i, then its must randomly select a process π whose initial state is consistent with input i. We then say that π *has input* i.

Hence a randomized distributed algorithm for function evaluation can be modelled as a random assignment of deterministic processes to processors where the random distribution for each assignment is determined by each processor's input value. We generalize this still further by permitting an *arbitrary* assignment of processes to processors. Let $\mathcal{A}$ be the set of all processes. A distributed *algorithm* α is a mapping from input values in a domain D to nonempty subsets of $\mathcal{A}$. The set $\alpha(i)$ is the collection of processes available to processors with input i. A sequence $\pi_{1,n} = \pi_1, \ldots, \pi_n$ corresponds to an assignments of processes to processors on rings of size n. The assignment is constrained only in that processor j with input i_j must be assigned a process in $\alpha(i_j)$.

This generalization from random to arbitrary assignments gives algorithms a nondeterministic attribute. Like conventional nondeterministic algorithms, an algorithm is said to evaluate $f(I)$ for $I \in D^n$ efficiently if for

some choice of process assignments consistent with I, the resulting computation has low complexity. Therefore lower bounds on the complexity of evaluating $f(I)$ address the complexity of the *best case.*

2.3 Function Evaluation

Let f be a cyclic function from D^n to $\{0,1\}$. Then f is the characteristic function of some $L \subseteq D^n$. Let $I = i_1, \cdots, i_n \in D^n$. Let $\pi_{1,n} = \pi_1, \ldots, \pi_n$ be a process sequence where $\pi_j \in \alpha(i_j)$, and let C be the computation of $\pi_{1,n}$. Then $\pi_{1,n}$ *evaluates f on input I* if C is an accepting computation when $I \in L$ and is a rejecting computation when $I \notin L$.

An algorithm α *evaluates f on input I* if for every $\pi_{1,n}$ with at least one initiator, where $\pi_j \in \alpha(i_j)$, $\pi_{1,n}$ evaluates f on input I. An algorithm α *evaluates f* if it evaluates f on input I for every $I \in D^n$.

The *complexity of an algorithm α* is *at least M* if there exists an input $I = i_1, \cdots, i_n \in D^n$ such that *for every* $\pi_{1,n} = \pi_1, \ldots, \pi_n$ with $\pi_j \in \alpha(i_j)$, the complexity of the computation associated with $\pi_{1,n}$ is at least M.

3 Lower Bound for Function Evaluation

Let f be any non-constant cyclic function of n variables, and let α be any distributed algorithm which evaluates f on a ring of size n where n is large. In this section we prove that there exists some input string I for which α requires $\Omega(n\sqrt{\log n})$ bits of communication to compute $f(I)$ even in the best case. Thus we conclude that the complexity of α is $\Omega(n\sqrt{\log n})$ bits.

The proof proceeds in two steps. Let α be an algorithm which evaluates f. The first step is to show that the claimed complexity applies whenever α is restricted to single initiator computations. The second step is a reduction which proves that any algorithm for function evaluation can be converted to an algorithm that works for any preassigned non-empty subset of initiators without entailing any significant additional complexity.

Theorem 3.1: Let $f: D^n \longrightarrow \{0,1\}$ be any non-constant cyclic decision function of n variables. Let α be any distributed algorithm which evaluates f on a ring of size n for any non-empty set P of initiators. Then there is some input $I \in D^n$ for which α requires $\Omega(n\sqrt{\log n})$ bits of communication to evaluate $f(I)$ when the size of P is 1.

Proof: The proof has two parts. We first show how to build a new computation of α from a given one that has low complexity and a single initiator.

Let $I = i_1, \cdots, i_n \in D^n$ and let $C = h_1, \cdots, h_n$ be a computation of $\pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$ and $\pi_1, \ldots, \pi_n$ has exactly one initiator, π_1. Suppose the complexity of C is less than $(n\sqrt{\log n})/3$. Let $r = |h_1|$. Because there is one initiator, each history has either r or $r-1$ messages. Each message requires at least one bit, therefore $n(r-1) < (n\sqrt{\log n})/3$ implying $r < \sqrt{\log n}/2$ for large enough n.

C is now collapsed by repeatedly applying lemma 2.1 and property 2.2 until all histories are distinct. Let $C' = h_{\beta_1}, \cdots, h_{\beta_l}$ be the resulting subsequence of C. Then by property 2.2, C' is a computation of $\pi' = \pi_{\beta_1}, \cdots, \pi_{\beta_l}$ with input $I' = i_{\beta_1}, \cdots, i_{\beta_l}$. By lemma 2.1, $l < n/\sqrt{\log n}$. By property 2.3, $(C')^r$ is a computation of $(\pi')^r = \pi^1_{\beta_1}, \cdots, \pi^1_{\beta_l}, \pi^2_{\beta_1}, \cdots, \pi^2_{\beta_l}, \cdots, \pi^r_{\beta_1}, \cdots, \pi^r_{\beta_l}$ with input $(I')^r$ of length $rl < n/2$. But each copy of π' in $(\pi')^r$ has exactly one initiator, $\pi^j_{\beta_1} = \pi_1$ for $1 \leq j \leq r$. Now imagine blocking all messages between $\pi^r_{\beta_l}$ and $\pi^1_{\beta_1}$. Then $\pi^r_{\beta_1}$ still receives $r-1$ messages and hence still has output history $h_{\beta_1} = h_1$.

Let $\gamma = \gamma_1, \cdots, \gamma_{n-rl}$ be any element of D^{n-rl} and $\tau = \tau_1 \cdots \tau_{n-rl}$ be any sequence with $\tau_j \in \alpha(\gamma_j)$. Consider the sequence $(\pi')^r\tau$ which has input $(I')^r\gamma$. Since any message generated by τ cannot influence the history of $\pi^r_{\beta_1}$ in $(\pi')^r\tau$ until after the r^{th} message, $\pi^r_{\beta_1}$ must have the messages of h_1 as its first r output messages. In particular, if h_1 is decisive, h_1 is the complete history of $\pi^r_{\beta_1}$.

In summary, this construction of collapsing, replicating and splicing can be applied to any $\pi_{1,n}$ and its input I whenever the computation C of $\pi_{1,n}$ is decisive and has complexity less than $n(\sqrt{\log n})/3$. The result is a new sequence $(I')^r$ of length $rl < n/2$ such that for any γ of length $n - rl > n/2$, there is a computation C'' which has input $(I')^r\gamma$ and some decisive history h_i occurs in both C and C''.

This construction is now used to find an input $I \in D^n$ for which the assumption that the complexity of any single initiator computation of α with this input is less than $(n\sqrt{\log n})/3$ leads to a contradiction.

Let $L \subset D^n$ be the language recognized by f. Let $d \in D$ and without loss of generality, assume $d^n \notin L$. (Otherwise consider the language $\bar{L}$.) Let $\omega = d^k\rho$ be an element of L such that k is maximum over all strings in L.

Case 1: Suppose $k \leq n/2$. Then let $I = \omega$. Since $\omega \in L$, C must be an accepting computation. Therefore there is some computation C'' containing an accepting history h_i for any γ. Let $\gamma = d^{n-rl}$. Then h_i is an accepting history when the input string has more than k consecutive d's. Contradiction.

Case 2: Suppose $k > n/2$. Then let $I = d^n$. Since $d^n \notin L$, C must be a

rejecting computation. Therefore there is some computation C'' containing a rejecting history h_i for any γ. Let $\gamma = d^{k-rl}\rho$. Then h_i is a rejecting history when the input is $\omega \in L$. Contradiction. ∎

The second step to the final result is a reduction.

Theorem 3.2: Let α be a distributed algorithm that evaluates some function f on rings of size n. Let $C(\alpha; I)$ denote the *bit* complexity of α on input I. Then there exists an algorithm $\hat{\alpha}$ that evaluates f on rings of size n for any preassigned non-empty subset P of initiators, and $\max_{P \neq \emptyset} C(\hat{\alpha}, I) \leq 4C(\alpha; I) + O(n)$.

Proof: Let P be some non-empty collection of processors on the ring designated as initiators. The algorithm $\hat{\alpha}$ proceeds as follows. Each processor $p \in P$ sends a package containing a "wake-up" message and the first message of α. Each processor $q \notin P$ waits until it receives a "wake-up" message together with all accumulated messages of α. It forwards as a package the wake-up message, its initial message of α if it has one, and all the appropriate responses of α to the package of messages received. When $p \in P$ receives its first package, it discards the "wake-up" message. For the remaining computation, every processor receives a package of messages of α and sends a corresponding package of the appropriate α responses until the algorithm terminates.

The bit communication of $\hat{\alpha}$ is just the bits of α plus n "wake-up" messages plus the packaging costs. But package delimiters can at most increase the cost of the computation of α by a factor of four, and the "wake-up" messages add an additional $O(n)$ bits. ∎

Note that the reduction holds for any non-empty set P of processors, and processors need have no knowledge of the size of P. (Of course the message complexity of $\hat{\alpha}$ might be substantially less than the message complexity of α, but the bit complexity remains comparable.) In particular, the reduction must hold for a set P of size one. But by theorem 3.1, there is no $\hat{\alpha}$ that has complexity less than $(n\sqrt{\log n})/3$ when the initiating set has size one.

Corollary 3.3: Let f be any non-constant cyclic decision function of n variables, and let α be any distributed algorithm which evaluates f on a ring of size n. Then the complexity of α is $\Omega(n\sqrt{\log n})$ bits.

Suppose f is any non-constant cyclic function with range S, and $s_1 \in S$ is one possible value of f. Then the function f' defined by

$$f'(x) = \begin{cases} 1 & \text{if } f(x) = s_1 \\ 0 & \text{otherwise} \end{cases}$$

is a non-constant cyclic decision function which can be computed at least as cheaply as f. Therefore the lower bound above actually applies to general function evaluation on a ring.

4 A Function that Achieves the Minimum Complexity

This section presents a non-constant boolean function f, which can be evaluated by a randomized algorithm in $O(n\sqrt{\log n})$ expected bits on a ring of size n.

Our algorithm for f relies on an algorithm for a simpler problem called *Solitude Detection*. Let a nonempty set of processors in a distributed system be distinguished. The problem of solitude detection is for every distinguished processor to determine whether there is one or more than one distinguished processor. In an algorithm for solitude detection, the initiators are precisely the distinguished processors. The complexity of solitude detection for a ring has been thoroughly studied in [2,3,4]. In particular, the following algorithm for solitude detection is a variation of a general algorithm that appears in [3].

Algorithm *SD*

Let m be the smallest integer such that $m \geq \sqrt{\log n}$ and m is relatively prime to n. It follows from the prime number theorem that $m = O(\log n)$ [9]. Messages are assumed to have two fields; *message type* and *message value*. For ease of description, five message types are used. However it is really only necessary to label *alarms* since other message types arrive in a fixed order and can thus be distinguished implicitly. The function *random*(x) is assumed to return an unbiased random coin toss of heads or tails and store the result in variable x.

```
Initiators:
  send(coin-toss, random(my-toss));
  round ←0; terminated ←false;
  while not terminated do
    receive message(type, value);
    case type of
      coin-toss:   if value = my-toss then
                     if round < log log n then
                       send(coin-toss, random(my-toss));
                       round ←round+1
                     else send(mod-count,1)
```

```
                    round ←0
                else more-than-one.
    mod-count:  if value = n mod m then
                    send(gap-count,1)
                else more-than-one.
    gap-count:  if value = long then
                    send(okay,–)
                else more-than-one.
    okay:       if round < √(log n) then
                    send(okay,–);
                    round ←round+1
                else alone ←terminated ←true.
    alarm:      more-than-one.
  procedure more-than-one:
    send(alarm,–); alone ←false; terminated ←true.

Non-initiators:
  repeat forever
    receive message(type, value);
    case type of
      coin-toss:    forward message.
      okay:         forward message.
      alarm:        forward message.
      mod-count:    send(mod-count, (value + 1) mod m).
      gap-count:    if value < n/√(log n) then
                        send(gap-count, value+1)
                    else send(gap-count, long )
```

Algorithm SD solves the solitude detection problem on unidirectional rings with expected communication complexity $O(n\sqrt{\log n})$ bits. The correctness and complexity proofs are only sketched here but can be found in detail in [3].

Correctness: When there is only one initiator,then it is readily confirmed that no alarms are generated and the algorithm terminates with alone assigned true for the sole initiator.

When there are $k \geq 2$ initiators, the "mod-count" messages ensure that an alarm is generated unless $k \geq m + 1$. The "gap-count" messages then ensure that at least one alarm is generated within every sequence of $\sqrt{\log n} \leq m$ initiators. Finally the "okay" messages ensure that alarms are forwarded to any initiators that have not already sent one, thus informing

all initiators of non-solitude.

Complexity: When there is one initiator, the coin tosses never produce an alarm so they account for $O(n\sqrt{\log n})$ bits. Counting mod m requires $O(n \log m) = O(n \log \log n)$ bits. A further $O\left(\frac{n}{\sqrt{\log n}} \log n\right) = O(n\sqrt{\log n})$ bits are used by the gap counter. Finally the okay messages require $O(n\sqrt{\log n})$ bits. Thus the complexity when there is one initiator is $O(n\sqrt{\log n})$ bits.

When there are two or more initiators, the probability is $\left(1 - \frac{1}{\log n}\right)$ that a given initiator will send an alarm before successfully sending and receiving $\log \log n$ pairs of matching coin tosses. Therefore the total expected bit complexity of "mod-count", "gap-count", and "okay" messages is $O\left(\frac{n}{\log n}(\log m + \log n + \sqrt{\log n})\right) = O(n)$. The expected cost of the coin tosses is $O(n)$ bits since each initiator sends $O(1)$ expected bits before sending an alarm. Alarms cost $O(n)$ bits always. So the total cost is $O(n)$ expected bits, when there are two or more initiators.

SD distinguishes between one and more than one initiator. Since nothing is computed if there are no initiators, SD cannot be trivially converted to a boolean function over all strings in $\{0,1\}^n$. We now construct a boolean function which, after a small amount of communication (at most $O(n \log \log n)$ bits), always leaves at least one processor in a distinguished state, and for some nonempty subset W of inputs, leaves exactly one processor in a distinguished state. The distinguished processors can then determine whether there is one or more than one distinguished processor by running SD. Hence the processors determine whether or not the input string is in W.

Let $\nu(n)$ be the smallest positive nondivisor of n. Note that $\nu(n) = O(\log n)$. Let $t = \lceil \log \nu(n) \rceil$ and $T = 2t + 2$. Assume $T < \nu(n)$. (Otherwise $\nu(n) \leq 10$ and a simpler approach results in a function that reduces to solitude detection in $n \cdot \nu(n) = O(n)$ bits.) Let $r = n/T$. Note that r is an integer.

A configuration of bits on a ring of size n is *well-formed* if it has the form $(1 \cdot 1 \cdot (0 \cdot \{0,1\})^t)^r$. Note that a well-formed configuration has a unique parse into r *blocks* of T bits of the form $1 \cdot 1 \cdot (0 \cdot \{0,1\})^t$. A block $1 \cdot 1 \cdot 0 \cdot b_{t-1} \cdot 0 \cdot b_{t-2} \cdot 0 \ldots 0 \cdot b_0$ *encodes* the integer whose binary representation is $b_{t-1}b_{t-2}\ldots b_0$. A well-formed configuration is *sequential* if successive blocks, (including block r followed by block 1), encode successive integers mod $\nu(n)$. A well-formed configuration is *almost sequential* if all but one pair of successive blocks encode a pair of successive integers mod $\nu(n)$. Since $\nu(n)$ does not divide n, sequential configurations do not exist. However,

almost sequential configurations are easily constructed.

Let f be the boolean function defined on strings $\omega \in \{0,1\}^n$ by:

$$f(\omega) = \begin{cases} 1 & \text{if } \omega \text{ is almost sequential} \\ 0 & \text{otherwise} \end{cases}$$

Theorem 4.1: Evaluation of f on a distributed ring reduces to solitude detection in $O(n \log \nu(n))$ bits.

Proof: We describe an algorithm which evaluates f assuming that there is a subroutine that solves solitude detection.

1. Each processor starts by sending its own input bit and forwarding $2T - 2$ more bits to its successor.

2. Each processor determines whether its sequence of $2T$ known bits is consistent with the configuration being well-formed. If this is so, it is *locally well-formed.* Each processor whose $2T$ known bits have the form $(1 \cdot 1 \cdot (0 \cdot \{0,1\}^t)^2$, determines if the configuration is *locally sequential*, that is, whether the consecutive blocks encode successive integers mod $\nu(n)$.

3. A processor is *distinguished* if either 1) it has determined that the configuration is not locally well-formed or 2) it has determined that the configuration is not locally sequential.

4. Distinguished processors initiate the solitude detection algorithm.

5. Upon termination of solitude detection, distinguished processors forward "function value is 1" to the next distinguished processor if solitude is confirmed and "function value is 0" otherwise.

It is easy to see that a configuration is well-formed if and only if it is everywhere locally well-formed. If a configuration is not well formed, it must be not locally well-formed in more than one place. Therefore there is one distinguished processor if and only if the configuration is almost sequential, and there is always at least one distinguished processor.

The first step requires the transmission of $n(2T - 1) = O(n \log \nu(n))$ bits. The last step requires $O(n)$ bits. Since the only other communication is due to the solitude detection algorithm, the reduction requires $O(n \log \nu(n))$ bits. ∎

Corollary 4.2: The non-constant cyclic boolean function f can be computed by a randomized distributed algorithm in expected complexity $O(n \log \nu(n)) + O(n\sqrt{\log n}) = O(n\sqrt{\log n})$ bits on a ring of size n.

5 Related Problems

The $\Omega(n\sqrt{\log n})$ lower bound for randomized function evaluation was proven tight by presenting a function that can be computed in $O(n\sqrt{\log n})$ expected bits. Similarly, [8] constructs a function that can be evaluated deterministically in $O(n \log n)$ bits. However both these examples of low complexity functions are somewhat contrived. The inherent complexity of more familiar boolean functions such as AND or OR is not addressed in this paper. When restricted to deterministic computation, such functions can be easily computed in $O(n^2)$ bits, and this is known to be optimal [5]. But AND and OR can be reduced to leader election in $O(n)$ bits. (In fact, for any regular set L, the function that recognizes L can be reduced to leader election in $O(n)$ bits of communication [7].) Since a leader can be elected on a ring of known size n in $O(n \log n)$ expected bits by a randomized algorithm [1], we conclude that these functions can be evaluated in $O(n \log n)$ expected bits using randomization as opposed to $O(n^2)$ deterministically.

We conjecture that $O(n \log n)$ expected bits is also optimal for AND and OR. However, any proof of this conjecture necessarily requires stronger techniques than those presented here, since there is a nondeterministic algorithm for AND that has complexity $O(n\sqrt{\log n})$ bits. This nondeterministic algorithm is achieved by first electing a leader, and then allowing the leader to circulate a single final message to compute the function value. Leader election takes advantage of exact knowledge of the ring size to run quickly in the best case. Each processor p_i first chooses a number $l_i \in [0, \nu(n) - 1]$ and sends it to its successor (where, again, $\nu(n)$ is the smallest nondivisor of n). If $l_i = l_{i-1} + 1$ then p_i drops out of contention for leadership. The remaining contenders run solitude detection. If solitude is confirmed then there is a leader. If it is not confirmed then the remaining contenders run any leader election algorithm. Since $\nu(n)$ does not divide n, there must remain at least one contender after the first exchange of messages. In the best case there will remain exactly one contender and it will be elected in $O(n \log \nu(n))$ bits for the elimination of contenders plus $O(n\sqrt{\log n})$ bits to confirm solitude. Thus a leader can be elected nondeterministically in $O(n\sqrt{\log n})$ bits.

Randomized function evaluation permits coin tosses only to decrease expected complexity. It is still required that the function be correctly evaluated upon termination and that termination occur with probability one for all possible inputs. These requirements could be weakened to *probabilistic* function evaluation — function evaluation that permits error with probability at most ϵ. The complexity of probabilistic solitude detection is known to be $\Theta(n \min(\log \nu(n) + \sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}, \log\log(1/\epsilon)))$ bits on rings of

known size n [3]. The function described in section 4 can be evaluated probabilistically using the same reduction as presented in this paper, followed by a probabilistic version of solitude detection. Hence the complexity of evaluating this function with confidence $1-\epsilon$ is $O(n \min(\log \nu(n) + \sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}))$. A lower bound of $\Omega(n \min(\sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}))$ is provided by a more elaborate version of the proof in this paper which incorporates error tolerance [6]. These bounds match to within a constant factor only if $\log \nu(n) = O(\sqrt{\log\log(1/\epsilon)})$. The complexity of probabilistic function evaluation remains an open question when this condition is not met.

References

[1] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, and D. Kirkpatrick. *The Bit Complexity of Randomized Leader Election on a Ring.* Technical Report 86-3, University of British Columbia, Vancouver B.C., 1986. submitted for publication.

[2] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. *Probabilistic Solitude Detection I: Rings Size Known Approximately.* Technical Report 87-8, University of British Columbia, 1987. submitted for publication.

[3] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. *Probabilistic Solitude detection II: Rings Size Known Exactly.* Technical Report 86-26, University of British Columbia, 1986. submitted for publication.

[4] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. Probabilistic solitude verification on a ring. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 161–173, 1986.

[5] C. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. In *Proc. 4th Annual ACM Symp. on Principles of Distributed Computing*, pages 196–203, 1985.

[6] L. Higham. Ph.d. thesis. in preparation.

[7] Y. Mansour and S. Zaks. On the bit complexity of distributed computations with a leader. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 151–160, 1986.

[8] S. Moran and M. Warmuth. Gap theorems for distributed computation. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 131–140, 1986.

[9] T. Nagell. *Introduction to Number Theory.* John Wiley and Sons Inc., New York, 1951.

[10] J. Pachl and D. Rotem. Notes on distributed algorithms in unidirectional rings. In *Proc. 1st International Workshop on Distributed Algorithms*, pages 115–122, 1985.