JOURNAL OF ALGORITHMS 12, 204–238 (1991)

Better Computing on the Anonymous Ring*

HAGIT ATTIYA†

Laboratory for Computer Science, MIT, Cambridge, Massachusetts 02139

AND

MARC SNIR

IBM T. J. Watson Research Center, Yorktown Heights, New York 10598

Received April 27, 1988; revised November 28, 1989

We consider a bidirectional ring of n processors, where processors are *anonymous*, i.e., are indistinguishable. In this model it is known that "most" functions (in particular XOR and orientation) have worst case message complexity $\Theta(n^2)$ for asynchronous computations, and $\Theta(n \log n)$ for synchronous computations. The average case behavior is different; an algorithm that computes XOR asynchronously with $O(n\sqrt{n})$ messages on the average is known. In this paper we give tight bounds on the average complexity of various problems. We show the following:

- An asynchronous deterministic algorithm that computes any computable function with $O(n \log n)$ messages, on the average (improving the $O(n\sqrt{n})$ algorithm). A matching lower bound is proven for functions such as XOR and orientation.
- An asynchronous probabilistic algorithm that computes any computable function with $O(n \log n)$ expected messages on any input, using one random bit per processor. A matching lower bound is proven.
- A Monte-Carlo asynchronous algorithm that computes any computable function with $O(n)$ expected messages on any input, using one random bit per processor, with fixed error probability $\varepsilon > 0$.

*A preliminary version of this paper appeared in "Proceedings, 3rd Aegean Workshop on Computing, Corfu, Greece, June/July 1988" (J. Reif, Ed.), pp. 329–338, Lecture Notes in Computer Science, Vol. 319, Springer-Verlag, New York/Berlin.

†Supported by ONR Contract No. N0014-85-K-0168, by NSF Contract No. CCR-8611442, and by DARPA Contract No. N00014-83-K-0125. Part of this work was done while this author was at the Department of Computer Science, Tel-Aviv University, Israel.

0196-6774/91 $3.00

- A synchronous algorithm that computes any computable function optimally in $O(n)$ messages, on the average.
- A synchronous probabilistic algorithm that computes any computable function optimally in $O(n)$ expected messages on any input, using one random bit per processor.
- Lower bounds on the complexity of Monte-Carlo algorithms that always terminate.

1. Introduction

We consider a distributed network of n processors, with a ring topology. Each processor is connected by a bidirectional communication channel to each of its two neighbors. The processors are indistinguishable from each other, and all execute the same algorithm (*anonymous ring*). In the *asynchronous* model of computation message transfer time is arbitrary (but always finite). In the *synchronous* model of computation message transfer time is fixed, and all processors are synchronized. These models have been studied by many authors. They allow us to understand the effect of symmetry on the complexity of distributed computations.

Attiya, Snir, and Warmuth [5] showed that deterministic algorithms for many problems in the asynchronous anonymous model require at least $\Omega(n^2)$ messages to be sent in the worst case; synchronous algorithms require $\Omega(n \log n)$ messages in the worst case. Two examples are the problem of computing the XOR of binary input values, and the problem of orienting a ring. Syrotiuk and Pachl [19] showed that this bound does not hold for the average case, and that these problems can be solved asynchronously with $O(n\sqrt{n})$ messages on the average (their algorithm is stated for the problem of orientation, but can be applied also for XOR).

Here we show that this result is not optimal. We prove the following results:

The most general problem, of collecting the input values, can be solved in the asynchronous model by a deterministic algorithm using $O(n \log n)$ messages on the average; it can be solved by a probabilistic algorithm that uses $O(n \log n)$ expected number of messages, on any input, with one random bit at each processor.

A matching $\Omega(n \log n)$ lower bound on the average complexity is shown for "nonlocal" problems, where the answer is not determined by a short substring of the inputs (in particular, for XOR and orientation). The lower bound makes use of the counting technique introduced by Bodlaender [7] for the average complexity of leader election. However, it is necessary to combine this technique with a method of "forcing" the transmission of messages, similar to those introduced in [5]. The lower bounds hold for bidirectional rings, with no assumptions on the algorithms. The algorithms

may be nonuniform, i.e., may depend on the ring size, n. The same $\Omega(n \log n)$ lower bound holds for asynchronous probabilistic algorithms. This lower bound was proven independently by Abrahamson, Alder, Higham, and Kirkpatrick [2, 13].

The input collection problem is solved on synchronous rings by an algorithm that uses $O(n)$ messages, on the average. A probabilistic algorithm solves the input collection problem with $O(n)$ expected number of messages on any input, using one random bit per processor.

These results have several interesting aspects. First, we show again that synchronous models are more powerful than asynchronous ones. Next, we provide an example where probabilistic methods provably reduce complexity; a surprising small amount of randomization is sufficient to achieve this result. Finally, we show that the "bad" inputs that force the use of n^2 (resp. $n \log n$) messages for deterministic asynchronous (resp. synchronous) algorithms are rare; these are input configurations with large amounts of symmetry [5].

The expected performance of probabilistic algorithms on a worst input is the same as the average case performance of deterministic algorithms, for input collection, and other "global" problems. This does not hold for all problems: We show that the AND function can be computed by a deterministic asynchronous algorithm with $O(n)$ messages, on the average, whereas any asynchronous probabilistic algorithm that computes AND requires $\Omega(n \log n)$ messages, in the worst case.

Finally, we examine the Monte-Carlo asynchronous algorithms. If a probability ε of error is tolerated, than the input collection problem can be solved with $O(n(1 + \log\log(2/\varepsilon)))$ expected number of messages on any input, which is linear for constant ε. This implies that a leader can be elected in a labeled asynchronous ring with $O(n)$ messages, and small, constant error probability ε. The algorithm is nonuniform, and depends on n, the ring size. Interestingly, a lower bound of $\Omega((1 - \varepsilon)n \log n)$ was proven by Pachl [15] for uniform leader election algorithms (algorithms that work on rigs of unknown size). Thus, there is a provable gap between uniform and nonuniform Monte-Carlo leader election algorithms.

A Monte-Carlo distributed algorithm may fail by deadlocking, or it may fail by arriving at a wrong answer. If deadlock is prohibited, then we show that any asynchronous algorithm that computes AND with probability of error at most ε uses $\Omega(n(\log n - \log\log(2/(1 - \varepsilon))))$ messages, in the worst case; this is $\Omega(n \log n)$ as long as $\varepsilon \ll 1 - 2^{-n}$. Thus, the reduction in message complexity for Monte-Carlo distributed algorithms is almost entirely due to the acceptance of some probability of deadlock.

The rest of this paper is organized as follows: In Section 2 we present the model and some preliminary results. Section 3 is dedicated to algorithms, for both the synchronous and the synchronous models. In Section 4

the lower bounds, for both models, are presented. We conclude, in Section 5, with discussion of the results.

2. Definitions

2.1. Deterministic Algorithms

Consider a system of n indistinguishable (anonymous) processors arranged on a bidirectional ring. We number the processors $1, \ldots, n$, for convenience; however, this numbering is external and is not available to the processors themselves.

Every processor i has two distinct links to its neighbors, left(i) and right(i). In an *oriented* ring we have left(i) = $i - 1$, and right(i) = $i + 1$. In a general ring the notions of "left" and "right" at different processors need not be consistent. We denote by O_i the *orientation* of processor i: $O_i = 1$ if right(i) = $i + 1$ (positive orientation), and $O_i = 0$ if right(i) = $i - 1$ (negative orientation). The value of O_i is not available to processor i; however, a processor may test if its orientation agrees with the orientation of its neighbors:

$$O_i = O_{\text{left}(i)} \qquad \text{if right(left}(i)) = i.$$

If processors may send messages on both communication links then the ring is *bidirectional*; an oriented ring in which processors may send messages only on one link (say *right*) is *unidirectional*.

An algorithm specifies the behavior of each processor, modeled as a state machine. The initial state of a processor is its input value. The state of a processor when in a halting state is its output value. In the *asynchronous* model processor transitions are message driven. A processor that is not in a halting state receives one message at a time; when it receives a message it possibly sends messages left and right and moves to a new state. Messages on a channel are delivered in the order they are sent, after an arbitrary (finite) delay. The first transition at each processor is initiated by a conceptual "start" message. In the *synchronous* model processor transitions are clock driven. At each cycle, a processor that is not in a halting state accepts messages sent by its neighbors at the previous cycle (if any), possibly send messages to its neighbors and moves to a new state.

We consider computation problems where inputs are from a finite alphabet, often binary; an output function f_i of all the inputs is computed at each processor. If the initial input on the ring is $I_1, \ldots, I_n$, then processor i halts with output $f_i(I_1, \ldots, I_n)$. We usually assume all processors compute the same function; e.g., all processors compute the XOR of the input values. A special case is the problem of *orientation*: We are given a bidirectional ring, unoriented (with null inputs); a consistent

orientation is desired. Formally, each processor i computes a binary output f_i, such that $f_i = f_j$ iff $O_i = O_j$.

It is impossible to solve certain problems (for example, XOR, orientation, and more) if the number of processors on the ring, the *size* of the ring, is unknown [5]. Hence, we assume a nonuniform model where a distinct algorithm may be used for each ring size n; we denote such algorithm by the subscript n.

The *complexity* of a synchronous algorithm $\mathscr{A}_n$ on input I, $\mathscr{C}(\mathscr{A}_n, I)$ is the number of messages sent in the computation of the algorithm on input I. The (*worst case*) *complexity* of an algorithm $\mathscr{A}_n$, $\mathscr{C}_{\max}(\mathscr{A}_n)$, is maximum of $\mathscr{C}(\mathscr{A}_n, I)$ over all inputs I; this is the maximum number of messages sent in any computation. The *average complexity*, $\mathscr{C}_{\text{aver}}(\mathscr{A}_n)$ is the average of $\mathscr{C}(\mathscr{A}_n, I)$ over all inputs. Asynchronous algorithms are nondeterministic; the computation may depend on the order of message arrival. We represent this by a *scheduler*. After each transition the scheduler selects the next message to be received. The *complexity* of an asynchronous algorithm $\mathscr{A}_n$ on input I, $\mathscr{C}(\mathscr{A}_n, I)$ is the number of messages sent in a computation of the algorithm on input I, against a worst scheduler. $\mathscr{C}_{\max}(\mathscr{A}_n)$, and $\mathscr{C}_{\text{aver}}(\mathscr{A}_n)$ are defined as above.

2.2. Probabilistic Algorithms

Deterministic algorithms are modeled by deterministic automata: in each state, and for each arriving message, there is a unique successor state. A *probabilistic* algorithm is modeled by a probabilistic automaton: for each state and incoming message there are several possible transitions, each associated with a probability. Transitions are independent.

A probabilistic algorithm solves a problem *with error* ε if for any input there is a probability $\geq 1 - \varepsilon$ that all processors halt with a correct answer to the problem. In particular, an *errorless probabilistic algorithm* always delivers the right answer. Note that a probabilistic algorithm may fail in two ways: it may terminate and deliver a wrong answer, or it may deadlock in a situation where there are no outstanding messages, but some processor has not yet halted.

The complexity of an asynchronous probabilistic algorithm $\mathscr{A}_n$ on input I, $\overline{\mathscr{C}}(\mathscr{A}_n, I)$, is the expected number of messages sent against a worst scheduler. The worst case complexity of a probabilistic algorithm, $\overline{\mathscr{C}}_{\max}(\mathscr{A}_n)$, and the average case complexity $\overline{\mathscr{C}}_{\text{aver}}(\mathscr{A}_n)$, are defined accordingly. The definition for synchronous algorithms is obvious.

2.3. Notation

Define the *k-neighborhood* of processor i to be the concatenation of the input values and orientations of the processors at most k apart from

processor i, relative to the orientation of i. This neighborhood is defined by the string of length $2k + 1$ $(O_{i-k}I_{i-k}),\ldots,(O_{i+k}I_{i+k})$. Since the neighborhood is defined relative to the orientation of processor i, the string $(\overline{O}_{i+k}I_{i+k}),\ldots,(\overline{O}_{i-k}I_{i-k})$ (i.e., the string obtained by reversing inputs and inversing all orientations) defines the same neighborhood (all indices are taken modulo n).

The $\lfloor n/2\rfloor$-neighborhood of a processor contains information on the entire ring configuration, relative to the location and orientation of the processor; this is the *most general information* about the ring a processor may acquire. For example, in a deterministic synchronous computation, the output of a processor depends only on its $\lfloor n/2\rfloor$-neighborhood. This implies that a function can be computed deterministically if and only if the output of processor i is a function of its $\lfloor n/2\rfloor$-neighborhood. Thus, the most general problem is that of computing for each processor its $\lfloor n/2\rfloor$-neighborhood. We call this the *input collection* problem. An algorithm that solves the input collection problem can be used to solve any problem that can be computed deterministically on a ring. We present such algorithm in the next section.

We denote by $|\sigma|$ the length of a string σ. We say that σ *appears cyclically* in τ if σ is the prefix of some cyclic shift of τ. For example, if i and j are two processors on a ring size n then the k-neighborhood of processor i appears cyclically in the $\lfloor n/2\rfloor$-neighborhood of processor j, for any $k \leq \lfloor n/2\rfloor$.

3. Upper Bounds

3.1. *Asynchronous Input Collection Algorithm*

Syrotiuk and Pachl [19] presented a deterministic asynchronous algorithm, solving the orientation problem using $O(n\sqrt{n})$ messages on the average. Here we show that their algorithm is not optimal, and that there exists a deterministic asynchronous input collection algorithm using $O(n \log n)$ messages, on the average.

3.1.1. *Informal Description*

For simplicity of description and analysis we first assume the ring to be unidirectional, and the input alphabet to be binary; the modifications required for the general case are described later.

The input collection problem can be solved with $O(n)$ messages once a leader has been elected on the ring: The leader initiates a message that circles the ring, first collecting all inputs, next distributing them to all

processors. We shall solve the input collection problem this way, first running a leader election process.

It is not always possible to elect a unique leader on an anonymous ring; if the ring is symmetric; e.g., if all processors have the same initial state, then this symmetry cannot be broken by a deterministic algorithm [3]. However, it is not essential for our purposes that a unique leader be elected. The algorithm is still correct if several leaders are elected; each of the elected leaders will distribute the inputs independently. We shall exhibit a leader election algorithm that ends by electing a constant number of leaders, on the average.

The leader election algorithm resembles the algorithm of Chang and Roberts [9]. This algorithm selects the processor with maximum id in a ring where processors have distinct ids. In this algorithm, each processor creates a message that travels around the ring, carrying its originator's id, until it "meets" a processor with a larger id. In the worst case at most $O(n^2)$ messages are sent during the computation. However, a message carrying the kth largest id travels average distance n/k. Altogether, $nH_n = O(n \log n)$ messages are sent on the average (H_n is the sum of the harmonic series with n elements).

In our model processors are identical; before the leader election algorithm can be run, ids must be computed. We label each processor by the number of consecutive ones to its left. Not all processes end up with distinct labels; we show that the expected number of processors with maximum labels is constant, and that the distribution of labels to processor still guarantees that the election process takes $O(n \log n)$ messages, on the average.

Thus, the algorithm consists of three conceptual phases:

1. Labeling.
2. Leader electron.
3. Input collection and distribution.

The actual algorithm given below combines phases two and three together: Inputs are collected by the messages used for leader election.

3.1.2. *Code for the Algorithm*

The algorithm uses the following variables (at each processor):

INPUT—Input value of the processor.
LABEL—Label created for this processor.

A message sent by the algorithm at the first phase consists of the Boolean input of the processor that initiated the message. A message sent at the second phase of the algorithm consists of a pair (*LABEL*,*SEG*),

```
Algorithm for processor i
  { First phase - label creation }
  LABEL := 0;
  send INPUT to right;
  repeat forever
        receive L from left;
        LABEL := LABEL + L;
        if LABEL = n then break;
        if INPUT = 1 then send L to right;
        if L = 0 then break
  end;

  { Second phase - leader election and input collection }
  send (LABEL, INPUT) to right;
  repeat forever
        receive (L, SEG) from left;
        if |SEG| = n then
        begin
          send (L, shift(SEG)) to right;
          break
        end;
        if L ≥ LABEL then
          send (L, SEG · INPUT) to right
          { else message is not forwarded }
  end.
```

FIG. 1. Asynchronous input collection algorithm.

where *LABEL* is the label of the processor that initiated the message, and *SEG* is a binary string obtained by concatenating the inputs on the ring segment traversed by the message.

The algorithm is described in Fig. 1; $\sigma \cdot \tau$ is the string obtained by concatenating σ and τ; shift(σ) is the function that shifts the string σ cyclically one position to the right.

3.1.3. *Correctness*

First phase. The following claim is easily proven, by induction on j: If there are j ones to the left of a processor, followed by a zero, then this processor will receive j one messages, followed by a zero message; the final value of $LABEL_i$ is j. If the ring contains only ones then each processor will send n one messages, receive n one messages, receive n one messages, and halt with $LABEL = n$. Thus each processor exits the first phase with *LABEL* being equal to the number of consecutive ones to the left of the processor.

Second phase. We say that a message is *full* it if has a segment of length n. It is easy to prove by induction that the successive messages sent by a processor have segments of increasing length, up to length n. Let L_0 be the maximum label. If a message (L_0, SEG), with $|SEG| = j < n$, is sent to a processor, then this processor has not yet received a full message, and has not halted; this processor receives the message and forwards it with a segment of length $j + 1$. It follows that a message with label L_0 is forwarded until it is full; this implies that at least one processor receives a full message and halts. A processor halts after it forwards a full message; hence if a processor halts then the next processor on the ring receives a full message and halts, too. Thus, all processors halt.

When processor i receives a full message (L,SEG) then SEG is the concatenation of the inputs at processors $i, i + 1, \ldots, n, 1, \ldots, i - 1$. This holds true when a message returns to the processor that initiated it, and stays true at later processors, since SEG is shifted.

If the ring is bidirectional and unoriented, one runs two versions of the algorithm in parallel, one in each direction. In addition to its input, a processor also depends to a collection message a bit that indicates what port the message is sent from. A processor halts after it has received and forwarded a full message in each direction. It is still the case that each processor i receives a message describing the ring configuration, i.e., the input and orientation of every processor, relative to the location and orientation of processor i. The average message complexity of the algorithm at most doubles.

3.1.4. *Extensions*

If the input alphabet has size $s > 2$, then we use a fixed encoding to assign a binary value to each letter (e.g., assume that that alphabet is $\{0, \ldots, s - 1\}$, then the ith letter of the alphabet is assigned value $i \bmod 2$). These binary values are used in the first phase of the algorithm to generate labels; in the second phase, the entire input is transferred. Note that if s is odd then zero and one are not equally likely to appear: A binary value of zero occurs with probability $\lceil s/2 \rceil / s$, and a value of one occurs with probability $\lfloor s/2 \rfloor / s$.

If the problem has no binary inputs, such as for orientation, then one can use the relative orientation of the processors to create binary values. We set $INPUT_i$ to one if the two neighbors of processor i have distinct orientations, to zero otherwise; i.e.,

$$INPUT_i = O_{i-1} \oplus O_{i+1}.$$

We have

$$O_{i+1} = INPUT_i \oplus O_{i-1}, \qquad i = 2, \ldots, n - 1,$$

so that

$$O_i = INPUT_{i-1} \oplus INPUT_{i-3} \oplus \cdots \oplus INPUT_{\alpha+1} \oplus O_\alpha,$$

where $\alpha = 1$ if i is odd, $\alpha = 2$ if i is even. Fix the values of O_1 and O_2; the last identities define a one-to-one correspondence between the values of $O_3, \ldots, O_n$ and the values of $INPUT_2, \ldots, INPUT_{n-1}$. Thus, if each orientation is equally likely to occur, then the tuple $INPUT_2, \ldots, INPUT_{n-1}$ is equally likely to assume each of the 2^{n-2} possible values. By symmetry, this holds true for any $n - 2$ consecutive locations.

Note, however, that not every binary configuration is obtained. An orientation cannot be distinguished from its mirror image by a distributed computation, so that the 2^n orientations can map into at most 2^{n-1} input configurations. In fact, if n is odd then only configurations with an even number of ones are created, each with probability 2^{n-1}; if n is even, then only a configuration with an even number of ones at the odd-numbered processors, and an even number of ones at the even-numbered processors, are created.

The algorithm described above does not "clean up" the ring: Messages not received may be left on the ring; this can be avoided while increasing the message complexity by at most a constant factor. Each "winning" message in the election algorithm performs one more full circle; all processors can count the number of winners. A processor halts at the next round after it has received that many input collection messages.

3.1.5. *The Average Number of Messages*

We shall use in our analysis the following combinatorial result.

THEOREM 3.1. *Let $\sigma = \sigma_1, \ldots, \sigma_k$ be a binary string. Let $x_1, x_2, \ldots$ be a sequence of independent binary random variables, that are one with probability p and zero with probability $q = 1 - p$. Let $W(\sigma)$ be the waiting time until the pattern σ occurs in the sequence $x_1, x_2, \ldots$:*

$$W(\sigma) = \min\{i : x_{i-k+1} = \sigma_1, \ldots, x_i = \sigma_k\}.$$

Let p_i be the probability of a match on the ith letter of the pattern σ: $p_i = p$ if $\sigma_i = 1$, $p_i = q$ if $\sigma_i = 0$. Then

$$E(W(\sigma)) \leq \frac{1}{p_k} + \frac{1}{p_{k-1}p_k} + \cdots + \frac{1}{p_1 \cdots p_k}.$$

Equality obtained only when $\sigma = 0^k$ or $\sigma = 1^k$. In the former case we obtain

$$E(W(0^k)) = \frac{1 - q^k}{pq^k},$$

and in the latter case,

$$E(W(1^k)) = \frac{1 - p^k}{qp^k}.$$

Proof. Define the following discrete process, with $k + 1$ states $S^0, \ldots, S^k$. The initial state is $s_0 = S^0$. If $s_{i-1} = S^{r-1}$ and $x_i = \sigma_r$ then $s_i = S^r$; otherwise $s_i = S^0$. This process corresponds to a pattern matching algorithm that searches for the pattern σ in the string $x_1, x_2, \ldots$, without backtracking. If a mismatch occurs, then pattern matching restarts at the next position in the string, with the first letter of the pattern. State S^k is reached only if an occurrence of the string σ is found: If $s_i = S^k$ then $x_i = \sigma_k, x_{i-1} = \sigma_{k-1}, \ldots, x_{i-k+1} = \sigma_1$. Let w_i be the expected waiting time until state S^i is reached for the first time. Then

$$E(W(\sigma)) \leq w_k.$$

The process just defined is a Markov process; when in state S^{i-1} a transition to state S^i occurs with probability p_i, and a transition to state S^0 occurs with probability $1 - p_i$. The waiting times fulfill the following equations:

$$w_0 = 0;$$
$$w_i = w_{i-1} + 1 + (1 - p_i)w_i, \qquad i = 1, 2, \ldots, k.$$

The equations are rewritten as

$$w_0 = 0;$$
$$w_i = \frac{w_{i-1} + 1}{p_i}, \qquad i = 1, 2, \ldots, k.$$

We obtain the solution

$$w_1 = \frac{1}{p_1},$$
$$w_2 = \frac{1 + p_1}{p_1 p_2},$$
$$\vdots$$
$$w_k = \frac{1 + p_1 + p_1 p_2 + \cdots + p_1 p_2 \cdots p_{k-1}}{p_1 p_2 \cdots p_k}$$
$$= \frac{1}{p_k} + \frac{1}{p_{k-1} p_k} + \cdots + \frac{1}{p_1 \cdots p_k}.$$

Assume that $\sigma_1 \neq \sigma_i$, for some $1 < i \leq k$. Then the nonbacktracking pattern matching process will fail to detect an occurrence of the pattern σ in the string

$$\sigma_1 \cdots \sigma_{i-1}\bar{\sigma}_i\sigma_2 \cdots \sigma_k = \sigma_1 \cdots \sigma_{i-1}\sigma_1\sigma_2 \cdots \sigma_k.$$

This implies a strict inequality $E(W(\sigma)) < w_k$. Conversely, the nonbacktracking pattern matching process detects all occurrences of a pattern 0^k or 1^k, and equality $E(W(\sigma)) = w_k$ holds. (If a 0 is found while searching for the pattern 1^k, then search for the full pattern *should* start anew from the next position in the string.) Substituting $p_i = p$ we obtain that

$$E(W(1^k)) = \sum_{i=1}^{k} \frac{1}{p^i} = \frac{1-p^k}{(1-p)p^k}.$$

The result for $E(W(0^k))$ is obtained by substituting $p_i = q$. □

Note that $W(1^k)$ is the waiting time for k consecutive successes in a sequence of Bernoulli trials. A derivation of the expectation of this waiting time can be found in [12, XIII. 7, Eq. (7.7)]. Let $r = \min(p, q)$. Then, since $p_i \geq r$,

$$w_k = \sum_{i=1}^{k} \prod_{j=i}^{k} \frac{1}{p_j} \leq \sum_{i=1}^{k} \frac{1}{r^i}.$$

We thus obtain

COROLLARY 3.2. *For any pattern σ of length k,*

$$E(W(\sigma)) \leq \max(E(W(0^k)), E(W(1^k))).$$

If $p < \frac{1}{2}$ then the pattern 1^k is the (unique) "worst case" pattern, with maximal waiting time; if $p > \frac{1}{2}$, then 0^k is the unique worst case pattern.

We shall analyze the algorithms under assumptions that subsume the various modifications introduced at the end of the last section. We assume that $INPUT_1, \ldots, INPUT_n$ are 0–1 valued random variables; $INPUT_i$ has value one with probability p, where $\frac{1}{3} \leq p \leq \frac{1}{2}$, and value zero with probability $q = 1 - p$; and any $n - 2$ consecutive random variables $INPUT_i, INPUT_{i+1}, \ldots, INPUT_{i-3}$ are independent. The assumption about p covers the case of non-binary alphabet, while the independence assumption allows us to treat the orientation problem.

First phase. A message initiated in the first phase is forwarded until it encounters a zero on the ring, or until it has done a full circle, if there are no zeros on the ring. Thus, the expected distance traversed by such

message is bounded by

$$\begin{aligned} E(W(0)|W(0) \le n-2) \cdot \mathrm{PR}[W(0) \le n-2] + n \cdot \Pr[W(0) > n-2] \\ \le E(W(0)) + 1 \\ = \frac{1}{q} + 1 \\ \le 3. \end{aligned}$$

Since exactly n messages are initiated in the first phase, it follows that the expected number of messages transmitted in this phase is $\le 3n$.

Second phase. Let $LABEL_i$ be the label created for processor i, and let X_i be the number of times the message sent by processor i is forwarded in this phase. Assume that $LABEL_i = k-1$. If k consecutive ones occur at locations $j-k, \ldots, j-1$, then $LABEL_i \ge k$ and processor j does not forward the message initiated by processor i. A processor halts after receiving a message of length n; hence, no message is forwarded more than $2n-1$ times. It follows, by Lemma 3.1, that for $1 \le k \le \log_{(1/p)} n$,

$$\begin{aligned} E(X_i|LABEL_i = k-1) &\le E(W(1^k)) + (2n-1) \\ &\quad \cdot \mathrm{PR}[W(1^k) \ge n-k-2] \\ &\le E(W(1^k)) + (2n-1)\frac{E(W(1^k))}{n-k-2} \\ &= \frac{1-p^k}{qp^k} \cdot \left(1 + \frac{2n-1}{n-k-2}\right) \\ &\le 4\frac{1-p^k}{qp^k}, \end{aligned}$$

for $n > 8$. Summing up over all the values of $LABEL_i$, we obtain that

$$\begin{aligned} E(X_i) &< \sum_{k=1}^{\log_{(1/p)} n} E(X_i|LABEL_i = k-1) \cdot \mathrm{PR}[LABEL_i = k-1] \\ &\quad + (2n-1) \cdot \mathrm{PR}[LABEL_i \ge \log_{(1/p)} n] \\ &\le \sum_{k=1}^{\log_{(1/p)} n} 4 \cdot \frac{1-p^k}{qp^k} \cdot qp^{k-1} + (2n-1) \cdot p^{\log_{(1/p)} n} \\ &< \frac{4}{p} \log_{(1/p)} n + \frac{2n-1}{n} \\ &< 8 \log n + 2. \end{aligned}$$

Since exactly n messages are initiated at this phase, it follows that the average number of messages transmitted is $O(n \log n)$.

The last result implies, in particular, that the expected number of leaders selected in this phase (defined as the number of messages that make a full circle) is $O(\log n)$. We show in Theorem 3.8, by a more accurate analysis, that this number is constant.

We sum up the results of this section in the following theorem.

THEOREM 3.3. *For any n there exists a deterministic input collection algorithm $\mathbf{IC}_n$, that works on the asynchronous anonymous ring of size n, and has average complexity $\mathscr{C}_{\text{aver}}(\mathbf{IC}_n) = O(n \log n)$.*

3.2. AND and Other Problems

The last theorem shows that any computable function can be computed with $O(n \log n)$ messages, on the average. In Section 4.2 we show that this result is optimal for problems such as XOR or orientation. For some other problems one can do better.

THEOREM 3.4. *For each n there is an asynchronous algorithm $\mathbf{AND}_n$ that computes the AND of n inputs on an anonymous ring of length n, with an average number of messages $\mathscr{C}_{\text{aver}}(\mathbf{AND}_n) = O(n)$.*

Proof. Each processor starts by sending to its right and left a message with its input value, and count one. Afterwards, it forwards the messages it receives, incrementing their count. A processor halts with output zero after it has sent a zero message; it halts with output one if it receives back a one message with count n (i.e., a one message that made a full circle). It is easy to check that the algorithm computes AND correctly. The expected distance traversed by a message is $\leq E(W(0)) = 1/q \leq 2$. Thus, the expected number of messages transmitted is linear. □

It is easy to see that this result is optimal (see Section 4.1). A similar algorithm can be used to compute any function whose value is determined by a small prefix. Let $f: \Sigma^n \to \Sigma$ be a shift invariant function. Let σ be a string of length k that determines the value of f, i.e., $f(\sigma \cdot \tau_1) = f(\sigma \cdot \tau_2)$, for any strings τ_1 and τ_2 of length $n - k$ (the string 0 plays this role for the AND function). We have

THEOREM 3.5. *The function f can be computed asynchronously on an oriented ring with $O(n3^k)$ messages, on the average.*

Proof. Each processor starts by sending an "input collection" message to its right; afterwards, it forwards messages it receives. Messages carry the input values of the processors they traversed. A processor halts after it forwards a message containing the substring σ, or when it receives a

message of length n, that has made a full circle. In each case, the processor can compute the function value when it halts.

The expected distance traversed by a message is equal to the expected waiting time for an occurrence of σ in a random string. By Corollary 3.2, since $\frac{1}{3} \leq p \leq \frac{1}{2}$,

$$E(W(\sigma)) \leq E(W(1k)) \leq \frac{1-p^k}{(1-p)p^k} \leq \frac{3(3^k-1)}{2}. \qquad \square$$

A similar algorithm works for nonoriented rings (we require then that f be invariant under shifts and reversals; this is a necessary condition for f to be computable on nonoriented, anonymous rings [5]). The AND algorithm is a particular case, for $k = 1$.

3.3. Asynchronous Probabilistic Algorithms

The deterministic input collection algorithm can be easily modified to yield a probabilistic algorithm, that solves the input collection problem in $O(n \log n)$ expected number of messages, on any input: Select a random bit at each processor, and use this bit to build labels. The expected number of messages sent by this algorithm does not depend on the input, and equals the average number of messages sent by the input collection algorithm of Section 3.1. We obtain:

THEOREM 3.6. *For any n there exists an errorless probabilistic input collection algorithm* $\mathbf{PIC}_n$ *for asynchronous rings of size n that uses on any input an expected number of messages* $\bar{\mathscr{C}}_{\max}(\mathbf{PIC}_n) = O(n \log n)$. *This algorithm uses a unique random bit at each processor.*

Thus, any solvable problem can be solved on an asynchronous anonymous ring with $O(n \log n)$ expected messages, using a unique random bit at each processor. In Section 4.3 we show this result is optimal for problems such as AND, XOR, and orientation. On the other hand, if a positive error probability is tolerated, then the expected number of messages can be reduced to $O(n)$.

THEOREM 3.7. *For any n and ε, such that $0 < \varepsilon < 1$ and $n > 16 \ln(2/\varepsilon) + 2$, there exists an asynchronous probabilistic input collection algorithm* $\mathbf{EPIC}_{n,\varepsilon}$ *for rings of size n that has error probability $\leq \varepsilon$ and expected number of messages* $\mathscr{C}_{\max}(\mathbf{EPIC}_{n,\varepsilon}) \leq n(12 + 2 \log\log(2/\varepsilon))$ *on any input.*

Proof. We use the same input collection algorithm as in Section 3.1, with one change: Let p be the probability of occurrence of 1 on the ring ($\frac{1}{3} \leq p \leq \frac{1}{2}$). Only processors with labels $\geq \lambda \log_{1/p}(n-2)$ participate in

the leader election and input collection phase; $\lambda = \lambda(n, q, \varepsilon)$ is a constant that will be defined later. As shown in Subsection 3.1.5, the expected number of messages sent in the first phase is $\leq 3n$. The algorithm succeeds if some processor participates. We have to show that the probability that no processor participates is at most ε, and that the expected number of messages sent in the second phase is $\leq n(9 + 2\log\log(2/\varepsilon))$.

We first show that the probability that no processor participates is at most ε. Let $m = n - 2$. Let ζ be the number of zeroes in the first m locations on the ring, and let $m_1, m_2, \ldots, m_\zeta$ be the successive locations of these zeroes. Let $X_1 = m_1$ and $X_i = m_i - m_{i-1}$, for $i = 2, \ldots, \zeta$. The label of processor m_i is $\geq X_i$ (the label of processor m_i is equal to X_i, if $i > 1$). Thus, it is sufficient to show that, with probability $\geq 1 - \varepsilon$,

$$\max_{1 \leq \zeta} X_i > \lambda \log_{1/p} m.$$

A random process with the same distribution can be defined as follows. Let W be the waiting time for a zero in a sequence of Bernoulli trials with probability p of one and probability $q = (1 - p)$ of zero. Let $W^{(1)}, W^{(2)}, \ldots$ be a sequence of independent random variables, equidistributed as W. Let

$$\nu = \max\left\{ j : \sum_{i \leq j} W^{(i)} \leq m \right\}.$$

Then $W^{(1)}, \ldots, W^{(\nu)}$ are distributed as $X_1, \ldots, X_\zeta$. Moreover, ν is distributed as the number of zeroes in a sequence of m Bernoulli trials. We have

$$\mathrm{PR}\left[\max_{i \leq \nu} W^{(i)} > \lambda \log_{1/p} m \right] \geq \Pr\left[\max_{i \leq mq/2} W^{(i)} > \lambda \log_{1/p} m \right] - \Pr\left[\nu < \frac{mq}{2} \right].$$

Raghavan [18] derives the following estimate for the lower tail of a Bernoulli distribution, using Chernoff's method [8]:

$$\mathrm{PR}[\nu - mq < -\gamma mq] < e^{-\gamma^2 mq/2}. \tag{1}$$

Taking $\gamma = \frac{1}{2}$, since $q \geq \frac{1}{2}$, it follows that

$$\mathrm{PR}\left[\nu < \frac{mq}{2} \right] \leq e^{-mq/8} \leq e^{-m/16}.$$

Thus, since $m > 16\ln(2/\varepsilon)$,

$$\mathrm{PR}\left[\nu < \frac{mq}{2}\right] \leq \frac{\varepsilon}{2}.$$

On the other hand,

$$\mathrm{PR}\left[W^{(i)} > \lambda \log_{1/p} m\right] = p^{\lambda \log_{1/p} m} = m^{-\lambda}$$

and

$$\mathrm{PR}\left[\max_{1 \leq mq/2} W^{(i)} > \lambda \log_{1/p} m\right] = 1 - (1 - m^{-\lambda})^{mq/2}.$$

Choosing

$$\lambda = -\frac{\log\left(1 - (\varepsilon/2)^{2/mq}\right)}{\log m} = -\log_m\left(1 - \left(\frac{\varepsilon}{2}\right)^{2/mq}\right),$$

we have that

$$(1 - m^{-\lambda})^{mq/2} = \varepsilon/2,$$

and

$$\mathrm{PR}\left[\max_{i \leq mp/2} W^{(i)} > \lambda \log_{1/p} m\right] = 1 - \varepsilon/2.$$

Thus, the probability that the algorithm succeeds is at least $1 - \varepsilon$.

The expected number of messages sent in the second phase is estimated as in Section 3.1.5. Let Y_i be the number of times the messages initiated by processor i is forwarded, then

$$\begin{aligned}
E(Y_i) &< \sum_{k=\lambda \log_{1/p} m}^{\log_{1/p} m} E(Y_i | LABEL_i = k-1) \cdot \Pr[LABEL_i = k-1] \\
&\quad + (2m+1)\Pr[LABEL_i > \log_{1/p} m] \\
&\leq \sum_{k=\lambda \log_{1/p} m}^{\log_{1/p} m} \frac{1-p^k}{qp^k} \cdot qp^{k-1} + (2m+1) \cdot p^{\log_{1/p} m} \qquad \text{by Lemma 3.1,} \\
&< \frac{(1-\lambda)\log_{1/p} m}{p} + \frac{2m+1}{m} \\
&< 3(1-\lambda)\log m + 2.5.
\end{aligned}$$

We have

$$m^{-\lambda} = 1 - (\varepsilon/2)^{2/mp}.$$

Note that for $x > 1$,

$$1 - 1/x < \ln x.$$

Substituting $x = (2/\varepsilon)^{2/mp}$, we obtain

$$n^{1-\lambda} = n\left(1 - \left(\frac{\varepsilon}{2}\right)^{2/mp}\right) \leq m \cdot \ln\left(\left(\frac{2}{\varepsilon}\right)^{2/mp}\right) = \frac{2}{p} \ln \frac{2}{\varepsilon}.$$

Thus,

$$\begin{aligned} E(Y_i) &< 3(1 - \lambda)\log m + 2.5 \\ &< 3 \log\left(\frac{2}{p} \ln \frac{2}{\varepsilon}\right) + 2.5 \\ &< 9 + 3 \log\log(2/\varepsilon). \end{aligned}$$

□

3.4. Synchronous Algorithms—Input Collection

The asynchronous input collection had three phases;

1. labeling;
2. leader election; and
3. input collection and distribution.

The first phase requires a linear number of messages, on the average. A more accurate analysis of the leader election process shows that a constant number of leaders are elected on the average; hence, the third phase takes, too, a linear number of messages, on the average. It is only the second, the leader election phase, that requires $\Omega(n \log n)$ messages. However, this phase can be avoided altogether in the synchronous model: We divide the input collection and distribution phase into $n + 1$ subspaces; at subspace i, which takes $2n$ cycles, only processors with label $n + 1 - i$ collect and distribute inputs. The algorithm stops as soon as a subspace with active processors occurs. Thus, only leaders collect and distribute inputs.

3.4.1. *Code for the Algorithm*

We describe the algorithm for unidirectional, oriented rings, with binary inputs. The algorithm is extended to nonoriented rings, and used to solve the orientation problem, as in the asynchronous case. The algorithm is described in Fig. 2.

```
Algorithm for processor i
  { First Phase }
  LABEL := 0;
  if INPUT = 1 then send message to right;
  for j := 1 to n do
      if received message from left then
      begin
        LABEL := LABEL + 1;
        if INPUT = 1 then send message to right
      end;

  { Second Phase }
  for i := n downto 0 do begin
      if LABEL = i then send INPUT to right;
      for j := 1 to 2n - 1 do
          if received message M then
            if |M| < n then send INPUT · M to right
            else begin  {|M| = n }
                 send shift(M) to right;
                 halt
            end
  end.
```

FIG. 2. Synchronous input collection algorithm.

3.4.2. *Analysis*

The correctness of the algorithm is now obvious. The total number of cycles used by it is at most $2n^2$. We shall show now that the expected number of messages sent is linear. The first phase is essentially identical to the first phase of the asynchronous input collection algorithm; the expected number of messages sent is $O(n)$.

A message is forwarded in the second phase at most $2n - 1$ times; we have to show that the expected number of messages initiated in this phase is constant. In order to do so we need the following lemma.

LEMMA 3.8. *Consider n independent sequences of Bernoulli trials, each with probability p for one (and probability $q = 1 - p$ for zero). Let $W^{(i)}$, $i = 1, \ldots, n$ be the waiting time for the first zero in sequence i. Let M_n be the number of sequences where the waiting time is maximum, i.e.,*

$$M_n = \left| \left\{ i : W^{(i)} = \max_{1 \le j \le n} W^{(j)} \right\} \right|.$$

Then

$$E(M_n) \le 1/p.$$

Proof. Rabin proves in [17] that $\Pr[M_n > 1] < \frac{1}{3}$, for $p = 0.5$; we use a similar technique to bound the expectation, for any p.

Assume k ones occur in the first round (this happens with probability $\binom{n}{k}q^{n-k}p^k$). If $k = 0$ then $M_n = n$; otherwise k sequences continue to wait for zero. This implies the recurrence relation

$$E(M_n) = nq^n + \sum_{k=1}^{n} \binom{n}{k} q^{n-k}p^k E(M_k).$$

We have $E(M_1) = 1$, so that the above formula simplifies to

$$\begin{aligned} E(M_n) &= nq^n + nq^{n-1}p + \sum_{k=2}^{n} \binom{n}{k} q^{n-k}p^k E(M_k) \\ &= nq^{n-1} + \sum_{k=2}^{n} \binom{n}{k} q^{n-k}p^k E(M_k) \\ &= nq^{n-1} + \sum_{k=2}^{n-1} \binom{n}{k} q^{n-k}p^k E(M_k) + p^n E(M_n). \end{aligned}$$

Thus,

$$E(M_n) = \left(nq^{n-1} + \sum_{k=2}^{n-1} \binom{n}{k} q^{n-k}p^k E(M_k) \right) / (1 - p^n).$$

Assume, by induction, that $E(M_i) \leq 1/p$, for $1 < i < n$, then

$$\begin{aligned} E(M_n) &\leq \left(nq^{n-1} + \frac{1}{p} \cdot \sum_{k=2}^{n-1} \binom{n}{k} q^{n-k}p^k \right) / (1 - p^n) \\ &= \left(nq^{n-1} + \frac{1}{p}(1 - q^n - nq^{n-1}p - p^n) \right) / (1 - p^n) \\ &= \frac{1}{p} - \frac{q^n}{p(1 - p^n)} \\ &< \frac{1}{p}. \end{aligned}$$

□

In order to cover the various extensions to the algorithm, me assume, as in the proof of Lemma 3.1, that $INPUT_1, \ldots, INPUT_n$ are 0–1 valued random variables; that $INPUT_i$ has value one with probability p, where $\frac{1}{3} \leq p \leq \frac{1}{2}$, and value zero with probability $q = 1 - p$; and that any $n - 2$ consecutive random variables $INPUT_i, INPUT_{i+1}, \ldots, INPUT_{i-3}$ are independent. As in the asynchronous case, the assumption about p allows us

to cover the case of non-binary alphabet, while the independence assumption allows us to treat the orientation problem.

The total number of messages initiated in the second phase is at most n. The probability that the ring contains only ones is 2^{-n}. This case contributes $n2^{-n} = O(1)$ to the expected number of messages initiated. Otherwise, the expected number of messages initiated equals the expected number of maximal length strings of consecutive ones. Let $m = n - 2$. Let ζ be the number of zeroes in locations $1, \ldots, m$ on the ring; let $m_1, m_2, \ldots, m_\zeta$ be the location of these zeroes; let $X_1 = m_1$ and $X_i = m_i - m_{i-1}$, $i = 2, \ldots, \zeta$. Let

$$\text{Max} = \left| \left\{ j : j \leq \zeta \text{ and } X_j = \max_{i \leq \zeta} X_i \right\} \right|.$$

It is sufficient, for our purposes, to show that $E(Max) = O(1)$. The problem can be restated as follows. Define $W^{(1)}, W^{(2)}, \ldots$ to be a sequence of independent, identically distributed random variables; $W^{(i)}$ is distributed as the waiting time for a zero in a sequence of Bernoulli trials with probability p for one (and probability $q = 1 - p$ for zero). Let

$$\nu = \max \left\{ j : \sum_{i \leq j} W^{(i)} \leq m \right\}.$$

Let

$$M_k = \left| \left\{ j : W^{(j)} = \max_{i \leq k} W^{(i)} \right\} \right|.$$

Then the random variable M_ν has the same distribution as Max.

To estimate the expectation of M_ν we show that ν is highly likely to be in an interval of size $2\sqrt{m} \log m$ around qm. Let w be the maximal value of $W^{(i)}$, for $1 \leq i \leq qm - \sqrt{m} \log m$. We show that it is highly likely that the value of $W^{(i)}$ equals or exceeds w at most a constant number of times in the interval for $qm - \sqrt{m} \log m < i < qm + \sqrt{m} \log m$. This implies that it is highly likely that M_ν, the number of maxima in the range $1 \leq i \leq \nu$ is larger by at most a constant term than $M_{mq - \sqrt{m} \log m}$, the number of maxima in the *fixed* range $1 \leq i \leq mq - \sqrt{m} \log m$. We can then use Lemma 3.8 to show that the expected number of such maxima is constant.

Let α be a constant in the range $\frac{1}{2} < \alpha < 1$; let c be a constant integer, to be defined later. Define the following events:

$$A_1 = [qm - \sqrt{m} \log m < \nu < qm + \sqrt{m} \log m];$$

$$A_2 = [\max_{i \leq qm - \sqrt{m} \log m} W^{(i)} > \alpha \log_{1/p} m];$$

$$A_3 = [|\{i : qm - \sqrt{m} \log m < i < qm + \sqrt{m} \log m \text{ and } W^{(i)} > \alpha \log_{1/p} m\}| \leq c].$$

Let $A = A_1 \cap A_2 \cap A_3$. If A occurs then at most c values of $W^{(i)}$, for $qm - \sqrt{m} \log m < i \leq \nu$, equal or exceed the maximal value of $W^{(i)}$, for i in the range $1 \leq i \leq qm - \sqrt{m} \log m$. Thus, if A occurs, then $M_\nu \leq M_{qm - \sqrt{m} \log m} + c$. Since $M_\nu \leq n$, it follows that

$$E(M_\nu) \leq E\left(M_{qm - \sqrt{m} \log m}\right) + c + n(1 - \mathrm{PR}[A]).$$

By Lemma 3.8, $E(M_{qm - \sqrt{m} \log m}) \leq 1/p \leq 3$. We shall be proving that

$$\mathrm{PR}[A_i] \geq 1 - \frac{1}{m}, \qquad \text{for } i = 1,2,3.$$

This implies that $\Pr[A] \geq 1 - 3/m$ and

$$E(M_\nu) \leq 3 + n \cdot \frac{3}{n-2} + c = O(1).$$

The random variable ν is distributed as the number of zeroes in a sequence of m independent trials, each with probability of success p. We use the Chernoff type estimate from [4] to bound the probability of the distribution tail. We have ([4, Proposition 2.4(b)], see also [11, p. 17]),

$$\mathrm{PR}[\nu - qm \geq \gamma qm] \leq e^{-qm\gamma^2/3}, \qquad \text{for } \gamma \leq 1.$$

Setting $\gamma = (\log m)/(q\sqrt{m})$ we obtain that

$$\mathrm{PR}\left[\nu > qm + \sqrt{m} \log m\right] \leq e^{-\log^2 m/3q} < 1/2m,$$

for sufficiently large m. For the lower tail we use the estimate given in Eq. (1). For $\gamma = (\log m)/(q\sqrt{m})$, as selected before, we obtain

$$\mathrm{PR}\left[\nu < qm - \sqrt{m} \log m\right] < e^{-\log^2 m/2q} < 1/2m,$$

for sufficiently large m. Thus, $\Pr[\bar{A}_1] < 1/m$. We have

$$\mathrm{PR}\left[W^{(i)} > \alpha \log_{1/p} m\right] = p^{\alpha \log_{1/p} m} = 1/m^\alpha.$$

Thus,

$$\begin{aligned}
\mathrm{PR}[\bar{A}_2] &= \left(1 - \frac{1}{m^\alpha}\right)^{qm - \sqrt{m} \log m} \\
&\leq \left(1 - \frac{1}{m^\alpha}\right)^{qm/2} \\
&\leq \left(e^{-1/m^\alpha}\right)^{qm/2},
\end{aligned}$$

since $1 - x < e^{-x}$ (for $x > 0$),

$$= e^{-(q/2)m^{1-\alpha}}$$
$$< 1/m$$

for sufficiently large m, since $1 - \alpha > 0$.

The probability that the event $[W^{(i)} > \alpha \log_{1/p} m]$ occurs exactly k times in the range $qm - \sqrt{m} \log m < i < qm + \sqrt{m} \log m$ is equal to

$$\binom{2\sqrt{m} \log m}{k} \left(\frac{1}{m^\alpha}\right)^k \left(1 - \frac{1}{m^\alpha}\right)^{2\sqrt{m} \log m - k}$$

It follows that

$$\mathrm{PR}[\bar{A}_3] = \sum_{k \geq c} \binom{2\sqrt{m} \log m}{k} \left(\frac{1}{m^\alpha}\right)^k \left(1 - \frac{1}{m^\alpha}\right)^{2\sqrt{m} \log m - k}.$$

Since $\alpha > \frac{1}{2}$, for sufficiently large m, $2\sqrt{m} \log m \cdot m^{-\alpha} < 1$, so that the terms in the sum are decreasing. Thus, we can use the first term to estimate the sum.

$$\sum_{k \geq c} \binom{2\sqrt{m} \log m}{k} \left(\frac{1}{m^\alpha}\right)^k \left(1 - \frac{1}{m^\alpha}\right)^{2\sqrt{m} \log m - k}$$
$$\leq (2\sqrt{m} \log m - c) \binom{2\sqrt{m} \log m}{c} \left(\frac{1}{m^\alpha}\right)^c \left(1 - \frac{1}{m^\alpha}\right)^{2\sqrt{m} \log m - c}$$
$$\leq 2\sqrt{m} \log m \cdot \frac{(2\sqrt{m} \log m)^c}{c!} \cdot \frac{1}{m^{\alpha c}}$$
$$= \frac{2^{c+1}}{c!} \cdot m^{(1/2-\alpha)c + 1/2} \cdot \log^{c+1} m$$
$$< \frac{1}{m},$$

for sufficiently large m, provided that $(\frac{1}{2} - \alpha)c + \frac{1}{2} < -1$, i.e., $c > 3/(2\alpha - 1)$. We take

$$c = \left\lfloor \frac{3}{2\alpha - 1} \right\rfloor + 1.$$

We have proven

THEOREM 3.9. *For each n there exists a deterministic synchronous input collection algorithm* $\mathbf{SIC}_n$ *for rings of size n that has average message complexity* $\mathscr{C}_{\mathrm{aver}}(\mathbf{SIC}_n) = O(n)$.

We can modify the deterministic input collection algorithm to obtain a probabilistic input collection algorithm that has average message complexity $O(n)$: Select a random bit at each processor, and use these bits to create labels in the first phase. Run the second phase as before. The expected number of messages sent does not depend on the input, and equals the average number of messages sent by the deterministic algorithm. We obtain

THEOREM 3.10. *For each n there exists an errorless probabilistic synchronous input collection algorithm* $\mathbf{PSIC}_n$ *for rings of size n that uses a unique random bit per processor, and has expected message complexity on any input* $\bar{\mathscr{C}}_{\max}(\mathbf{PSIC}_n) = O(n)$.

4. LOWER BOUNDS

As shown in the next subsection, linear lower bounds are easily derived for nontrivial distributed computations. This implies that the linear synchronous algorithms are optimal. We next derive an $\Omega(n \log n)$ lower bound on the average complexity of deterministic asynchronous algorithms for "global" problems, such as XOR and orientation. The lower bound makes use of a counting technique introduced by Bodlaender [7, Sect. 2.3.5], together with an adversary technique from [5].

A probabilistic algorithm that makes only fair coin tosses, and has a fixed bound on the number of coin tosses, can be viewed as a process whereby each processor first selects a random binary string of fixed length, next runs a deterministic algorithm which uses both the original input and the random string as input. We first show that an arbitrary probabilistic algorithm can be transformed in an algorithm of the above form (with a fixed number of coin tosses), with a small change in error probability and message complexity. We can then derive lower bounds on the (worst case) complexity of probabilistic algorithms from bounds on the (average) complexity of deterministic algorithms.

4.1. Synchronous Algorithms

It is easy to see that a deterministic algorithm that computes a nonconstant function must use at least $n/2$ messages on the average; indeed, either processors with input zero send messages before they receive any, or processors with input one do so. In either case $n/2$ messages are sent, on the average, on the first cycle. Let f be a nonconstant Boolean function, and assume that $f(\sigma_1, \ldots, \sigma_{n-1}, 0) \neq f(\sigma_1, \ldots, \sigma_{n-1}, 1)$. Then, to compute f correctly, each processor $i, 1 \leq i \leq n-1$, has to receive a message either in the computation with input $\sigma_1, \ldots, \sigma_{n-1}, 0$, or in the

computation with input $\sigma_1, \ldots, \sigma_{n-1}, 1$ (otherwise it will halt with the same output in both computations). An errorless probabilistic algorithm that computes f sends at least $(n-1)/2$ messages on one of these two inputs. Thus, the synchronous upper bounds are optimal, up to a constant factor.

4.2. Deterministic Asynchronous Algorithms

Lower bounds for asynchronous computations are proven using as adversary a suitable scheduler. We use a simple *synchronizing scheduler* that keeps the computation as symmetric as possible. This scheduler delivers messages in *cycles*. All processors start the execution at cycle one; all messages sent at cycle i are received at cycle $i+1$.

LEMMA 4.1. *Under the synchronizing scheduler, the state of a processor after i cycles depends only on its i-neighborhood.*

If the algorithm is deterministic then each value of the neighborhood determines a unique state for the processor. In a probabilistic algorithm then the neighborhood determines the probability distribution for the processor state.

DEFINITION 4.1. A function f defined on Σ^n is m-global if, for any string σ of length $|\sigma| < m$, there exist two strings τ_1 and τ_2 such that $|\tau_1| = |\tau_2| = n$, σ appears cyclically both in τ_1 and in τ_2, and $f(\tau_1) \neq f(\tau_2)$.

The function f is m-global if its value cannot be determined from the values of $< m$ consecutive inputs. The XOR function is n-global; its value can not be determined from the value of less than n inputs. If f is an m-global function computed on a ring, then the value of f can not be determined from the value of a $(\lfloor m/2 \rfloor - 1)$-neighborhood. More generally, we say that a computation problem is *m-global* if on any initial input, the output at a processor cannot be determined from the value of a $(\lfloor m/2 \rfloor - 1)$-neighborhood. We have

LEMMA 4.2. *The orientation problem is* $(n/2)$*-global.*

Proof. Consider an input configuration (i.e., a sequence of orientations) $O_1, \ldots, O_n$; let $m < n/2$, and consider an initial configuration $O_1, \ldots, O_m \bar{O}_m, \ldots, \bar{O}_1, 1, \ldots, 1$. In this configuration processors $\lceil m/2 \rceil$ and $\lceil 3m/2 \rceil$ have the same $(m/4)$-neighborhoods. However, they have distinct orientations and, hence, different outputs. It follows that the output of processor $\lceil m/2 \rceil$ is not determined by its $(m/4)$-neighborhood. The same argument works for any processor. □

The proof of the following lemma is immediate from the definition of the synchronizing adversary.

LEMMA 4.3. *Consider a computation of an algorithm under the synchronizing scheduler. If no message is sent in that computation at cycle i, then no transition occurs, and no message is sent at any cycle j, for $j > i$.*

COROLLARY 4.4. *Let $\mathscr{A}_n$ be an asynchronous algorithm that solves an m-global problem on rings of size n. Consider a computation of $\mathscr{A}_n$ under the synchronizing scheduler. Then a message is sent by some processor at each cycle i, for $i = 1, \ldots, \lfloor m/2 \rfloor$.*

Proof. If no message is sent at cycle i, then, by Lemma 4.3, no further messages are sent, and no state transitions occur; the output of each processor is its state at cycle $i + 1$, which is determined by its $(i + 1)$-neighborhood (by Lemma 4.1). This implies that $i + 1 > \lfloor m/2 \rfloor$. □

Let $\mathscr{A}_n$ be an algorithm for rings of size n. Let $S_k(\mathscr{A}_n)$ be the set of k-neighborhoods that cause a message to be generated by the processor with that neighborhood at the kth cycle, when the algorithm $\mathscr{A}_n$ is executed with the synchronizing scheduler.

Our proof uses a counting argument similar to that used by Bodlaender [7, Sect. 2.3.5] to show that the sets $S_k(\mathscr{A}_n)$ are large. However, since rings are bidirectional, the set of neighborhoods initiating a message is not closed under the prefix operation (as in the corresponding multisets in [7 or 16]). Thus, a more delicate counting argument is needed.

THEOREM 4.5. *Let $\mathscr{A}_n$ be an asynchronous deterministic algorithm that solves an m-global problem on rings of size n. Let σ be a string of length k that describes the configuration of k successive processors on a ring, where k/n and $k < m$; let r be an integer such that $2r + 1 \leq k$. Then $S_r(\mathscr{A}_n)$ contains a string σ' which appears cyclically in σ.*

Proof. Look at the configuration $C = \sigma^{n/k}$. By Corollary 4.4, a message is sent by some processor at each cycle $r, r = 1, \ldots, \lfloor k/2 \rfloor$, of the computation of $\mathscr{A}_n$ on C under the synchronizing adversary. The r-neighborhood of this processor is the required string. □

COROLLARY 4.6. *Let $\mathscr{A}_n$, r, and k be as in the previous lemma. Let s be the size of the input alphabet. Then*

$$|S_r(\mathscr{A}_n)| \geq s^{2r+1}/k.$$

Proof. Each string of length $2r + 1$ appears cyclically in at most $ks^{k-(2r+1)}$ of the s^k distinct strings of length k. The claim follows from the previous lemma. □

THEOREM 4.7. *Let $\mathscr{A}_n$ be an asynchronous deterministic algorithm that solves an m-global problem on rings of size n. Let $1 = d_0 < d_1 < d_2 < \cdots < d_l = n$ be the sequence of divisors of n, ordered in increasing*

order. Then

$$\mathscr{C}_{\text{aver}}(\mathscr{A}_n) \geq n \sum_{1<d_i\leq m} \frac{\lceil d_i/2\rceil + \lceil d_{i-1}/2\rceil}{d_i}.$$

Proof. Let σ be a random input of length n. Let τ be the r-neighborhood of a fixed processor in σ, where $d_{i-1} < 2r+1 \leq d_i \leq m$. According to Corollary 4.6 (since there are s^{2r+1} strings of length $2r+1$),

$$\Pr[\tau \in S_r(\mathscr{A}_n)] \geq \frac{(s^{2r+1})/d_i}{s^{2r+1}} = \frac{1}{d_i}.$$

Thus, the expected number of messages sent at cycle r on input σ is $\geq n/d_i$, for $2r+1 \leq d_i \leq m$. The total expected number of messages is obtained by summing

$$\mathscr{C}_{\text{aver}}(\mathscr{A}_n) \geq \sum_{1<d_i\leq m} \sum_{d_{i-1}<k\leq d_i,\, k \text{ odd}} \frac{n}{d_i} = n \sum_{1<d_i\leq m} \frac{\lceil d_i/2\rceil - \lceil d_{i-1}/2\rceil}{d_i}.$$

□

COROLLARY 4.8. *Let $n = 2^k$, then for any asynchronous deterministic algorithm $\mathscr{A}_n$ solving an $\Omega(n)$-global problem on rings of size n, $\mathscr{C}_{\text{aver}}(\mathscr{A}_n) = \frac{1}{4}n \log n - O(n)$.*

Proof. The numbers $2^1, 2^2, \ldots$ are all dividers of n. We obtain

$$\mathscr{C}_{\text{aver}}(\mathscr{A}_n) \geq n \sum_{i=1}^{\log(cn)} \frac{2^{i-1} - 2^{i-2}}{2^i} = \frac{1}{4}n(\log n + \log c).$$ □

Since XOR and orientation are $\Omega(n)$-global, it follows:

COROLLARY 4.9. *Let $n = 2^k$, then for any asynchronous deterministic algorithm $\mathscr{A}_n$ computing XOR or orientation on rings of size n, $\mathscr{C}_{\text{aver}}(\mathscr{A}_n) = \frac{1}{4}n \log n - O(n)$.*

4.3. Probabilistic Algorithms

We assume henceforth that algorithms use a fixed, finite alphabet for inputs, outputs, and messages (the size of the alphabet may depend on n, the ring size). However, we do not restrict the number of processor states to be finite. Also, probabilistic choices with infinitely many branches are allowed. A probabilistic algorithm may execute finitely many steps, and yet have infinitely many states, if infinitely branching probabilistic choices are allowed. Also, a probabilistic algorithm may use finitely many states, have finite (expected) complexity, and yet allow (with probability zero) for infinite executions. We say that an algorithm is *finite* if there is a fixed upper bound on the number of processor states and on the number of

transitions (this bound may depend on n). A probabilistic algorithm is *binary* if all the transition probabilities are of the form $p/2^q$, for integer p and q; each probabilistic choice can be made by tossing a fair coin a fixed number of times.

LEMMA 4.10. *Let $\mathscr{A}_n$ be a probabilistic asynchronous algorithm with error probability ε such that $\bar{\mathscr{C}}_{\max}(\mathscr{A}_n) = c < \infty$. Then, for any $\delta > 0$, $\mathscr{A}_n$ can be replaced by a finite, binary probabilistic asynchronous algorithm $\widetilde{\mathscr{A}_n}$ that has error probability $\leq \varepsilon + \delta$ and worst case complexity $\bar{\mathscr{C}}_{\max}(\widetilde{\mathscr{A}_n}) \leq c + \delta$.*

Proof. We first take care that no processor execute more than a fixed number L of transitions. We modify the algorithm so that after L transitions a processor moves to a new "shutdown" state. A processor in the shutdown state sends a shutdown message and halts in an arbitrary state. Each processor, upon receiving a shutdown message, forwards the message and halts in an arbitrary state. Since each transition is associated with a message, the expected number of processors that execute L transitions, and generate a shutdown message is no more than c/L; the expected number of shutdown message transfers is at most nc/L. The worst case complexity of the modified algorithm is at most $c + nc/L$. The probability of error is at most $\varepsilon + c/L$.

We next round transition probabilities to multiples of $1/2^q$, where q is an integer to be defined later. This can be done so that no transition probability is modified by more than $1/2^q$. Consider a fixed input I, and schedule H. Assume that before rounding the algorithm executed with probability α a sequence of transitions $\mathscr{T} = T_1, \ldots, T_j$, $j \leq nL$. Then, after rounding, the algorithm executes the sequence $\mathscr{T}$ of transitions with probability α', where $\alpha(1 - 1/2^q)^j \leq \alpha' \leq \alpha(1 + 1/2^q)^j$. Thus, the new algorithm reaches a correct final state with probability at least $(1 - \varepsilon - c/L)(1 - 1/2^q)^{nL}$; the expected total number of transitions is at most $(c + nc/L)(1 + 1/2^q)^{nL}$. It follow that the error probability and the expected complexity can be bounded by $\varepsilon + \delta$ and $c + \delta$, respectively, for a suitable choice of q and L.

With the new algorithm each processor executes at most L transitions. Each nonzero transition probability is $\geq 2^{-q}$. Thus, at most 2^q distinct states can be reached from any state. If there are m distinct initial states (input values) then each processor can reach at most

$$m \cdot \sum_{i=1}^{L} 2^{qi} \leq m2^{q(L+1)}$$

distinct states. □

The transformation given in the last lemma maps an errorless probabilistic algorithm into a finite, binary algorithm that has an arbitrarily small, but nonzero error probability. However, if the original algorithm always terminates (is deadlock-free) then the derived algorithm always terminates.

Let $\mathscr{A}_n$ be a finite, binary algorithm that always terminates. There is a fixed upper bound on the number of probabilistic choices done by the algorithm, and each can be replaced by a fixed number of coin tosses. Therefore, we can replace an execution of $\mathscr{A}_n$ by a computation whereby each processor first chooses independently a random binary string of fixed length q, next runs a deterministic algorithm $\mathscr{A}_n^d$ (with input set $\Sigma \times \{0,1\}^q$). The expected complexity of $\mathscr{A}_n$ on an input $a_1, \ldots, a_n$ equals the average complexity of $\mathscr{A}_n^d$ over all inputs $a_1 s_1, \ldots, a_n s_n, s_i \in \{0,1\}^q$. If $\mathscr{A}_n$ has error probability ε, then $\mathscr{A}_n^d$ yields the "correct" answer on a fraction $\geq 1 - \varepsilon$ of these inputs. Thus, we can obtain bounds on the expected complexity of a probabilistic algorithm on one input from the average complexity of a deterministic algorithm.

DEFINITION 4.2. A function f defined on Σ^n is m-global on input $\sigma \in \Sigma^n$ if, for any string τ of length $|\tau| < m$ that appears cyclically in σ, there exists a string $\sigma' \in \Sigma^n$ such that τ appears cyclically in σ' and $f(\sigma) \neq f(\sigma')$.

The function f is m-global on σ if its value cannot be determined from the value of $< m$ consecutive inputs, for the input assignment σ. More generally, we say that a computation problem is *m-global on input $\sigma \in \Sigma^n$* if, on input σ, the output at a processor cannot be determined from the value of its $(\lfloor m/2 \rfloor - 1)$-neighborhood.

A function that is m-global, is m-global on any input. Thus, the function XOR is n-global on any input; the orientation problem is $(n/2)$-global on any input. The AND function is n-global on the input configuration $1, \ldots, 1$: The value of the AND function cannot be determined from the values of $m < n$ inputs, if these inputs happen to be all equal to one. Note that any other input configuration contains at least one zero, which determines the output value.

THEOREM 4.11. *Let $n = 2^m$. Let f be a Boolean function defined on rings of size n that is $\Omega(n)$-global on the input $1, \ldots, 1$. Let $\mathscr{A}_n$ be a probabilistic asynchronous algorithm that computes f with error ε, and always terminates. Then*

$$\bar{\mathscr{C}}_{\text{aver}} > \frac{n}{8}\left(\log n - \log\log\left(\frac{2}{1-\varepsilon}\right)\right) - O(n).$$

Proof. By Lemma 4.10, it is sufficient to prove the claim in the case where $\mathscr{A}_n$ is a finite, binary algorithm. Let $\mathscr{A}_n^d$ be the deterministic algorithm associated with $\mathscr{A}_n$, with input set $\{0,1\} \times \{0,1\}^q$. Let $\Sigma = \{0,1\}^q$. Since the function f is $\Omega(n)$-global there exists a constant $c > 0$ such that, for sufficiently large n, cn consecutive inputs with value one do not determine the value of f. Let k be an integer such that k/n and $k \leq cn$. Let $S_0^k \subseteq \Sigma^k$ be the set of strings $\sigma = \sigma_1, \ldots, \sigma_k$ such that a processor with neighborhood $1\sigma_1, \ldots, 1\sigma_k$ halts after $\leq \lfloor k/2 \rfloor$ steps with output zero, when the algorithm $\mathscr{A}_n^d$ is executed under the synchronizing scheduler. Let S_1^k be similarly defined, for output one, and let $S^k = S_0^k \cup S_1^k$. Let $\alpha_k = |S^k|/2^{kq}$ be the fraction of strings of length k that are in S^k. Then either $|S_0| \geq \alpha_k 2^{qk-1}$, or $|S_1| \geq \alpha_k 2^{qk-1}$.

Assume that the former holds. Let $a = a_1, \ldots, a_n$ be an input, where $a_1 = \cdots = a_{cn} = 1$, such that $f(a) = 1$. Then $\mathscr{A}_n^d$ yields a wrong answer on any input of the form $a_1\sigma_1, \ldots, a_n\sigma_n$, where $\sigma_1, \ldots, \sigma_{cn}$ contains a substring $\tau \in S_0^k$ (some processor halts with output zero). The probability that a randomly chosen string $\sigma_1, \ldots, \sigma_{cn}$ contains a substring from S_0^k is at least.

$$1 - \left(1 - \frac{|S_0|}{2^{qk}}\right)^{\lfloor cn/k \rfloor} \geq 1 - \left(1 - \frac{\alpha_k}{2}\right)^{cn/k-1}.$$

Thus, we must have

$$1 - (1 - \alpha_k/2)^{cn/k-1} \leq \varepsilon,$$

which implies that

$$\alpha_k \leq 2\left(1 - (1 - \varepsilon)^{k/(cn-k)}\right).$$

The same conclusion is drawn if $|S_1| \geq \alpha_k 2^{qk-1}$.

Let $\sigma = \sigma_1, \ldots, \sigma_k \notin S^k$. Consider the execution of the algorithm $\mathscr{A}_n^d$ under the synchronizing scheduler, with input $(1\sigma_1, \ldots, 1\sigma_k)^{n/k}$. Some processor has not halted after $\lfloor k/2 \rfloor$ computation cycles, and the algorithm always terminates. This implies that some message was sent at each of the first $\lfloor k/2 \rfloor$ cycles. Define, as in Subsection 4.2, $S_r(\mathscr{A}_n^d)$ to be the set of r-neighborhoods that cause a message to be sent at cycle r, when $\mathscr{A}_n^d$ is executed under the synchronizing scheduler. If $\sigma = \sigma_1, \ldots, \sigma_k \notin S^k$, then a string from $S_r(\mathscr{A}_n^d)$ appears cyclically in σ, for each $r \leq (k-1)/2$. This implies that, if $2r + 1 \leq k$, then

$$|S_r(\mathscr{A}_n^d)| \geq \frac{|\Sigma^k - S^k|}{k2^{q(k-2r-1)}} = \frac{(1 - \alpha_k)2^{q(2r+1)}}{k}.$$

Using the same argument as in the proof of Theorem 4.7, we obtain that the average number of messages sent by the algorithm $\mathscr{A}_n^d$ on inputs of the form $1\sigma_1, \ldots, 1\sigma_n$ is at least

$$\sum_{i=1}^{\log(cn)} \sum_{2^{i-1}<k\le 2^i,\, k \text{ odd}} \frac{(1-\alpha_2 i)n}{2^i} \ge \frac{n}{4} \sum_{i=1}^{\log(cn)} (1-\alpha_2 i)$$

$$\ge \frac{n}{4} \sum_{i=1}^{\log(cn)} \left(2(1-\varepsilon)^{2^i/(cn-2^i)} - 1\right). \quad (2)$$

Assume that

$$k \le \frac{cn \log(4/3)}{\log(4/3) + \log(1/(1-\varepsilon))} = \beta.$$

Then

$$2(1-\varepsilon)^{k/cn-k} - 1 \ge \frac{1}{2}.$$

Thus, we can bound the sum in Eq. 2 by

$$\frac{n}{4} \sum_{i=1}^{\lfloor \log \beta \rfloor} \frac{1}{2} = \frac{n}{8} \lfloor \log \beta \rfloor$$

$$\ge \frac{n}{8} \left(\log n - \log \log \left(\frac{2}{1-\varepsilon} \right) - O(1) \right). \qquad \square$$

The last theorem implies that a probabilistic algorithm that computes the AND function, or the XOR function, on a ring of length $n = 2^m$, always terminates, and has error probability ε, uses at least $(n/8) \cdot (\log n - \log \log(2/(1-\varepsilon)) - O(1))$ expected number of messages on the input $1, \ldots, 1$. A similar argument shows that the lower bound holds for the orientation problem. This implies that an errorless algorithm that solves any of the problems AND, XOR, or orientation uses at least $(n/8) \cdot (\log n - O(1))$ expected number of messages on a worst input.

The lower bound is $\Omega(n \log n)$, for error probability $\varepsilon < 1 - 2^{-\lambda n}$, $\lambda < 1$. This is to be contrasted to the upper bound of $O(n(1 + \log \log(2/\varepsilon)))$ we obtained in Theorem 3.7, where deadlock was allowed. Thus, the reduction in message complexity from $O(n \log n)$ to $O(n)$ achieved by

the Monte-Carlo algorithm of Theorem 3.7 is almost entirely due to the acceptance of the possibility for deadlock. Indeed, that algorithm either deadlocks, or yields the correct answer.

The lower bound of Theorem 4.11 is, essentially, optimal. Interestingly, it is possible to compute any computable function with $O(n)$ messages on a ring of odd length n, with a probability of success $n2^{-n}$, and no deadlock. Let $n = 2m + 1$; in [6] it is shown that ring configurations of the form $(01)^m1$ can be recognized with $O(n)$ messages by an asynchronous algorithm. There are such ring configurations, and in each such ring there are two "special" processors, where two consecutive ones occur. This implies that two leaders can be elected on a ring of odd length n with probability $n2^{-n}$, by a finite asynchronous probabilistic algorithm that uses $O(n)$ messages, and never deadlocks. Such leaders can collect and distribute inputs using $O(n)$ messages. It follows that we can run a finite asynchronous, probabilistic input collection algorithm that never deadlocks, succeeds with probability $n2^{-n}$, and uses $O(n)$ messages. Moran and Warmuth [14] generalize this construction to strings of arbitrary size, showing that a leader can be elected with probability $n2^{-n}$ using $O(n \log^* n)$ messages.

5. Concluding Remarks

We have shown in Theorem 3.5 that whenever there exists a string of length less than $\log\log n$ that determines the value of a function f, then f can be computed in $O(n \log n)$ messages, on the average. On the other hand, when the shortest such string has length cn, then the $n \log n$ lower bound applies. This leaves an open gap between $\log\log n$ and cn.

The algorithm presented in Theorem 3.7 can be used to find the maximum in a labeled ring with error probability $\leq \varepsilon$, in $O(n)$ messages. This would seem to contradict the $\Omega((1 - \varepsilon)n \log n)$ lower bound given by Pachl [15] for this problem. However, our algorithm depends on n, the ring size; the lower bound of Pachl applies only to algorithms that do not depend on n. Thus, our result in Theorem 3.7 together with the lower bound of Pachl, show a provable gap in message complexity between uniform distributed (probabilistic) algorithms and nonuniform ones.

Since input collection can be reduced to leader election, the lower bound of Corollary 4.8 implies that any election algorithm that elects on an asynchronous anonymous ring a constant average number of leaders has message complexity $\Omega(n \log n)$. The input collection algorithm can be used to obtain matching upper bounds. Also, any Monte-Carlo algorithm that does such election with probability at least $1 - \varepsilon$ and always terminates has message complexity $\Omega(n(\log n - \log\log(2/(1 - \varepsilon))))$.

The reduction technique from probabilistic algorithms to deterministic ones is very simple, yet very powerful. For example, it is trivial to show, using this method, that the $\Omega(n \log n)$ lower bound on the average complexity of a deterministic asynchronous leader choosing algorithm [7, 10, 16] implies an $\Omega(n \log n)$ lower bound on the average complexity of an errorless probabilistic asynchronous leader choosing algorithm.

The reduction of worst case analysis of probabilistic algorithms to average case analysis of deterministic algorithms is similar to the analysis used by Yao in [20]. Note, however, two important differences: We consider only one probability distribution on inputs, namely that inputs are equiprobable; and choices in a randomized algorithm are done independently by each processor. This is the reason, for example, that the average deterministic complexity of AND in the asynchronous model is $O(n)$, whereas the worst case complexity of a probabilistic asynchronous algorithm is $\Omega(n \log n)$.

The bit complexity of the asynchronous input collection algorithm can be improved by separating the leader election phase from the input collection and distribution phase. We modify the leader election phase by replacing the segment field of messages used in this phase by a counter that encodes the length of the segment (the counter counts the number of nodes traversed by the message). Each elected leader then sends a message around the ring that first collects all input values, next distributes them.

The labeling phase uses n messages of constant size, and each message travels on the average a constant distance; the average bit complexity is $O(n)$. Input collection and distribution by one leader takes $O(n^2)$ bit transfers (assuming binary inputs). The expected number of leaders participating in this phase is constant, so that this phase has average bit complexity $O(n^2)$. The expected distance traversed by a message in the leader election phase is $O(\log n)$, so that the expected number of bit transfers per message is $O(\log^2 n)$; the average bit complexity of the leader election phase is $O(n \log^2 n)$. Thus, the algorithm uses $O(n^2)$ bit transfers, on the average. Simple information transfer arguments show that input collection requires $\Omega(n^2)$ bit transfers, on the average. Thus, the modified algorithm has optimal average bit complexity. The number of bit transfers is reduced to $O(n \log^2 n)$ for functions such as XOR, orientation, SUM, etc., where it is sufficient to use messages of size $O(\log n)$ in the input collection and distribution phase. Similarly, input collection can be done probabilistically with $O(n^2)$ bit complexity, and XOR and orientation with $O(n \log^2 n)$ bit complexity, in the worst case. [1] contains results concerning the bit complexity of probabilistic algorithms for computing various functions on the ring.

Finally, note that the synchronizing scheduler we use to prove lower bounds for asynchronous rings is very simple, and input independent; it merely mimics a synchronous computation (compare with the complex malicious scheduler used in [10]). This scheduler keeps the computation as symmetric as possible. Here, as in [5], the lower bounds reflect the cost of breaking symmetry.

ACKNOWLEDGMENTS

The authors thank Michael Benor, Larry Carter, Don Coppersmith, Rafi Hassin, Jan Pachl, Prabhakar Raghavan, and Larry Stockmeyer for their useful suggestions. We also thank the anonymous referees for their thorough work.

REFERENCES

1. K. ABRAHAMSON, A. ADLER, L. HIGHAM, AND D. KIRKPATRICK, Randomized function evaluation on a ring, *in* "Proceedings, 2nd International Workshop on Distributed Algorithms, Amsterdam, Netherlands, July 1987" (J. van Leeuwen, Ed.), pp. 324–331, Lecture Notes in Computer Science, Vol. 312, Springer-Verlag, New York/Berlin, 1987.
2. K. ABRAHAMSON, A. ADLER, L. HIGHAM, AND D. KIRKPATRICK, "Probabilistic Evaluation of Common Function on Rings of Known Size," Technical Report 88-15, Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, June 1988.
3. D. ANGLUIN, Local and global properties in networks of processors, *in* "Proceedings, 12th Annual ACM Symp. on Theory of Computing, 1980," pp. 82–93.
4. D. ANGLUIN AND L. G. VALIANT, Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. System Sci.* **19** (1979), 155–193.
5. H. ATTIYA, M. SNIR, AND M. WARMUTH, Computing on the anonymous ring, *J. Assoc. Comput. Mach.* **35**, No. 4 (1988), 845–875.
6. H. ATTIYA, M. SNIR, AND M. WARMUTH, "Computing on the Anonymous Ring," Technical Report UCSC-CRL-85-3, University of California Santa Cruz, Nov. 1985.
7. H. BODLAENDER, "Distributed Algorithms: Structure and Complexity," Ph.D. thesis, University of Utrecht, 1986.
8. H. CHERNOFF, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations, *Ann. Math. Statist.* **23** (1952), 493–509.
9. E. CHANG AND R. ROBERTS, An improved algorithm for decentralized extrema-finding in circular configurations, *Comm. ACM* **22** (1979), 281–283.
10. P. DURIS AND Z. GALIL, Two lower bounds in asynchronous distributed computation, *in* "Proceedings, 28th Annual IEEE Symposium on Foundations of Computer Science, 1987," pp. 326–330.
11. P. ERDÖS AND J. SPENCER, "Probabilistic Methods in Combinatorics," Academic Press, New York, 1974.
12. W. FELLER, "An Introduction to Probability Theory and Its Applications," 3rd ed., Vol. 1, Wiley, New York, 1968.
13. L. HIGHAM, "Randomized Distributed Computing on Rings," Ph.D. thesis, Department of Computer Science, University of British Columbia, October 1988.
14. S. MORAN AND M. K. WARMUTH, Gap theorems for distributed computation, *in* "Proceedings, 5th ACM Symposium on Principles of Distributed Computations, 1986," pp. 141–150.

15. J. PACHL, A lower bound for probabilistic distributed algorithm, *J. Algorithms* **8** (1987), 53–65.
16. J. PACHL, E. KORACH, AND D. ROTEM, A new technique for proving lower bounds for distributed maximum-finding algorithms, *J. Assoc. Compt. Mach.* **31** (1984), pp. 905–918.
17. M. RABIN, N-process synchronization by $4 \cdot \log_2 N$-valued shared variable, *J. Compt. System Sci.* **25** (1982), 66–75.
18. P. RAGHAVAN, "Randomized Rounding and Discrete Ham-Sandwich Theorems: Provably Good Algorithms for Routing and Packing Problems," Report UCB/CSD 87/312, Computer Science Division, University of California Berkeley, July 1986.
19. V. SYROTIUK AND J. PACHL, Average complexity of a distributed orientation algorithm, *in* "Proceedings, 2nd International Workshop on Distributed Algorithms, Amsterdam, Netherlands, July 1987" (J. van Leeuwen, Ed.), pp. 332–336, Lecture Notes in Computer Science, Vol. 312, Springer-Verlag, New York/Berlin, 1987.
20. A. C. YAO, Probabilistic computations: Toward a unified measure of complexity, *in* "18th Annual Symposium on Foundations of Computer Science, 1977," pp. 222–227.