1. INTRODUCTION

A number of processors \(n\) are arranged in a ring configuration, in which each processor is connected by communication channels to 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.

In [ASW] it is shown 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 showed that these problems can be solved asynchronously with \(O(n \sqrt{n})\) messages on the average. Here we show that their result can be further improved. 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 holds for bidirectional rings and for nonuniform algorithms that depend on the ring size. The same \(\Omega(n \log n)\) lower bound holds for asynchronous probabilistic algorithms.

The input collection problem is solved on synchronous rings by an algorithm that uses \(O(n)\) messages, on the average. This algorithm can be used to elect a leader in a labeled ring with \(O(n)\) 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 provide an example where probabilistic methods provably reduce complexity; a surprisingly small amount of randomization is sufficient to achieve this result.

The worst case performance of probabilistic algorithms is not always equal to the average case performance of deterministic algorithms: 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 Monte-Carlo asynchronous algorithms. If a probability \(p\) of error is tolerated, than the input collection problem can be solved with \(O(n (1 + \log \log(1/p)))\)
expected number of messages. This implies that a leader can be elected in a labeled asynchronous ring with $O(n)$ messages, and small, constant error probability $p$. The algorithm is nonuniform, and depends on $n$, the ring size. Interestingly, a lower bound of $\Omega((1-p)n \log n)$ is given [P] for uniform leader election algorithms (algorithms that work on rings of unknown size).

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 $p$ uses $\Omega(n (\log n - \log(1/(1-p)))$ messages, in the worst case. Thus, the reduction in message complexity for Monte-Carlo distributed algorithms is almost entirely due to the acceptance of some probability of deadlock.

2. DEFINITIONS AND PRELIMINARY RESULTS

Consider a system of $n$ processors, numbered $1, \ldots, n$, on a ring (the numbering is not available to the processors). Processor $i$ has links to its two neighbors, left$(i)$ and right$(i)$. We denote by $O_i$ the orientation of processor $i$: $O_i = 1$ if right$(i) = i+1$, $O_i = 0$ otherwise. We consider computation problems where inputs are Boolean and a function of the inputs, such as XOR or AND, is computed at each processor. A special case is the problem of orientation: 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 and orientation) if the number of processors on the ring is unknown [ASW]. Hence we assume that an algorithm is designed for a specific ring size $n$; we denote such algorithm by the subscript $n$.

The complexity of a synchronous algorithm $A_n$ on input $I$, $C(A_n, I)$ is the number of messages sent in a computation on input $I$. The (worst case) complexity of an algorithm $A_n$, $C_{\text{max}}(A_n)$, is maximum of $C(A_n, I)$ over all inputs $I$. The average complexity, $C_{\text{avg}}(A_n)$ is the average of $C(A_n, I)$ over all inputs. Asynchronous algorithms are non-deterministic; the computation may depend on the order messages are forwarded. We represent this by a scheduler. After each transition the scheduler selects the next message to be received. The complexity of an asynchronous algorithm $A_n$ on input $I$, $C(A_n, I)$ is the number of messages sent in a computation of the algorithm on input $I$, against a worst scheduler. $C_{\text{max}}(A_n)$, and $C_{\text{avg}}(A_n)$ are defined as above.

Deterministic algorithms are modeled by deterministic automata; a probabilistic algorithm is modeled by a probabilistic automaton, with probabilistic transitions. A probabilistic algorithm solves a problem with error $p$ if for any input there is a probability $\geq 1-p$ that all processors halt with a correct answer to the problem. In particular, an errorless probabilistic algorithm always delivers the right answer. The complexity of an asynchronous probabilistic algorithm $A_n$ on input $I$, $\overline{C}(A_n, I)$, is the expected number of messages sent against a worst scheduler. The worst case complexity of a probabilistic algorithm, $\overline{C}_{\text{max}}(A_n)$, and the average case complexity, $\overline{C}_{\text{avg}}(A_n)$, are defined accordingly. The definition for synchronous algorithms is obvious.

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 $i$. The $\lceil n/2 \rceil$-neighborhood of a processor contains information on the entire ring configuration, relative to the location and orientation of the processor. The input collection problem consists of computing for each processor its $\lceil n/2 \rceil$-neighborhood. An algorithm that solves the input collection problem can be used to solve any problem that can be computed on a ring.
3. UPPER BOUNDS

3.1. Asynchronous Input Collection Algorithm

For simplicity of description and analysis we first assume the ring to be unidirectional. 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. It is not always possible to elect a unique a leader on an anonymous ring [An]. However, 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 [CR]; a processor with maximum label is elected leader. In this algorithm, each processor creates a message that travels around the ring, carrying its originator’s id, until it “meets” a processor labeled with a larger id. A message carrying the id $k$ travels distance $k$ on the average. Altogether, $O(n \log n)$ messages are sent on the average. In our model processors are identical; before the leader election algorithm can be run, id’s must be computed. We shall label each processor by the number of consecutive ones to its left. Thus, the algorithm consists of three conceptual phases:

1. Labeling.
2. Leader election.
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. The algorithm is described below; $s \cdot t$ is the string obtained by concatenating $s$ and $t$; $\text{shift}(s)$ is the function that shifts the string $s$ cyclically one position to the right.

Algorithm for processor $i$

\begin{align*}
\text{LABEL} & := 0; \\
\text{send } \text{INPUT} \text{ to right;} \\
\text{repeat forever} \\
& \quad \text{receive } L \text{ from left;} \\
& \quad \text{LABEL} := \text{LABEL} + L; \\
& \quad \text{if } \text{LABEL} = n \text{ then break;} \\
& \quad \text{if } \text{INPUT} = 1 \text{ then send } L \text{ to right;} \\
& \quad \text{if } L = 0 \text{ then break} \\
\end{align*}

\begin{align*}
\text{send (LABEL ,INPUT ) to right;} \\
\text{repeat forever} \\
& \quad \text{receive (L ,SEG ) from left;} \\
& \quad \text{if } |SEG| = n \\
& \quad \text{then begin} \\
& \quad \quad \text{send (L ,shift(SEG )) to right;} \\
& \quad \quad \text{break} \\
& \quad \quad \text{end;} \\
& \quad \text{if } L \geq \text{LABEL} \text{ then} \\
& \quad \quad \text{send (L ,SEG :INPUT ) to right} \\
& \quad \quad \{\text{else message is not forwarded}\} \\
\end{align*}

If the ring is bidirectional and unoriented, one runs two versions of the algorithm in parallel, one in each direction. The message complexity of the algorithm at most doubles. If the problem has no binary inputs, such as for orientation, then one can compute an “input bit” by comparing the orientation of a processor relative to the orientation of its left neighbor. If each initial orientation is equally probable, then each string of $n$
zeroes and ones is equally likely to obtain.

**Lemma 3.1:** Let $W_k$ be the waiting time until $k$ consecutive successes occur in a sequence of Bernoulli trials, with success probability $p = 0.5$. Then

$$E(W_k) = 2^{k+1} - 2.$$

**Proof:** see [Fe, XIII.7, eq. (7.6)]. □

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 zeroes on the ring. Thus, the expected distance traversed by such message is bounded by

$$E(W_1 I W_1 < n) \cdot \Pr[W_1 < n] + n \Pr[W_1 \geq n] < 2 + n \cdot 2^{-(n-1)} = O(1).$$

Since exactly $n$ messages are initiated in the first phase, it follows that the expected number of messages transmitted in this phase is $O(n)$.

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 the 2nd phase. Assume that $LABEL_i = k$. If $k + 1$ consecutive ones occur at locations $j-k-2, \ldots, j-1$ then $LABEL_j > k$ and processor $j$ does not forward the message initiated by processor $i$.

No message is forwarded more than $2n-1$ times. It follows, by Le. 3.1, that for $1 \leq k \leq \log n$

$$E(X_i | LABEL_i = k - 1) < E(W_k | W_k < n - k) \cdot \Pr[W_k < n - k] + (2n - 1) \cdot \Pr[W_k \geq n - k]$$

$$\leq E(W_k) + (2n - 1)E(W_k)/(n - k) \leq 2^{k+1} - 2 + (2n - 1)(2^{k+1} - 2)/(n - k) \leq 2^{k+3}.$$

Summing up over all the values of $LABEL_i$ we obtain that

$$E(X_i) < \sum_{k=1}^{\log n} E(X_i | LABEL_i = k - 1) \cdot \Pr[LABEL_i = k - 1] + (2n - 1) \cdot \Pr[LABEL_i \geq \log n]$$

$$\leq \sum_{k=1}^{\log n} 2^k + 32^{-k} + (2n - 1)2^{1-\log n} < 8\log n + 4.$$

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

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

**Theorem 3.2:** For each $n$ here exists a deterministic input collection algorithm $IC_n$, that works on the asynchronous anonymous ring, and has average complexity $C_{\text{avg}}(IC_n) = O(n \log n)$.

### 3.2. AND and Other Problems

The last theorem shows that any computable function, can be computed with $O(n \log n)$ messages, in the average. In §4 we show this result is optimal for problems such as XOR or orientation. For some other problems one can do better.

**Theorem 3.3:** For each $n$ there is an asynchronous algorithm $\text{AND}_n$ that computes the AND of $n$ inputs on an anonymous ring of length $n$, with an average number of messages $C_{\text{avg}}(\text{AND}_n) = O(n)$.

**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 the algorithm computes AND correctly and that the expected distance traversed by a message is constant. □

A similar algorithm can be used to compute any function which value is determined by a small prefix. Let \( f : \Sigma^n \rightarrow \Sigma \) be a shift invariant function. Let \( s \) be a string of length \( k \) that determines the value of \( f : f (s \cdot t_1) = f (s \cdot t_2) \), for any strings \( t_1, t_2 \) of length \( n-k \). We have

**Theorem 3.4:** The function \( f \) can be computed asynchronously on an oriented ring with \( O(nk 2^k) \) messages, on the average.

**Proof:** See [AS]. □

A similar result holds for nonoriented rings (we require then that \( f \) be invariant under shifts and reversals). The AND algorithm is a particular case, for \( k = 1 \).

### 3.3. Asynchronous Probabilistic Algorithms

The input collection algorithm can be easily modified to yield a probabilistic algorithm, that solves the input collection problem in \( O(n \log n) \) expected 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 to the average number of messages sent by the input collection algorithm of §3.1. We obtain:

**Theorem 3.5:** For each \( n \) there exists an errorless probabilistic input collection algorithm \( \text{PIC}_n \) that uses on any input an expected number of messages \( C_{\text{max}}(\text{PIC}_n) = O(n \log n) \). 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 §4 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 3.6:** Let \( p \) be a fixed positive number. There exist an asynchronous probabilistic input collection algorithm \( \text{EPIC}_n \) with error probability \( \leq p \) and expected number of messages \( C_{\text{max}}(\text{EPIC}_n) = O(n (1 + \log \log(1/p))) \) on any input.

**Proof:** See [AS]. □

### 3.4. Synchronous Algorithms - Input Collection

Consider the asynchronous input collection algorithm, with the basic three phases: (1) labeling; (2) leader election; and (3) input collection and distribution. The 1st 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 3rd phase takes, too, a linear number of messages, on the average. The 2nd phase can be avoided altogether in the synchronous model: We divide the input collection and distribution phase into \( n+1 \) subphases; at subphase \( i \), which takes \( 2n \) cycles, only processors with label \( n+1-i \) collect and distribute inputs. The algorithm stops as soon as a subphase with active processors occur. Thus, only leaders collect and distribute inputs.

We describe below the algorithm for unidirectional, oriented rings. The algorithm is extended to nonoriented rings, and used to solve the orientation problem, as in the asynchronous case. The total number of cycles used by it is at most \( 2n^2 \). We show that the expected number of messages sent is linear.
Algorithm for processor $i$

\begin{align*}
\{ \text{First Phase} \} & \quad \{ \text{Second Phase} \} \\
LABEL & := 0; \quad \text{for } i := n \text{ downto } 0 \text{ do} \\
\text{if } INPUT & = 1 \text{ then send message to right;} \quad \text{begin} \\
\text{for } j & := 1 \text{ to } n \text{ do} \quad \text{if } LABEL = i \text{ then send } INPUT \text{ to right;} \\
\quad \text{if received message from left then} & \quad \text{for } j := 1 \text{ to } 2n-1 \text{ do} \\
\quad \begin{align*}
\text{begin} & \quad \text{if received message } M \text{ then} \\
\quad \quad \text{if } \mid M \mid < n & \quad \begin{align*}
\text{then send } INPUT&M \text{ to right;} \\
\text{else } \mid M \mid = n & \quad \begin{align*}
\text{begin} & \quad \text{send } \text{shift}(M) \text{ to right;} \\
\text{halt} & \quad \text{end} \\
\text{end} & \quad \text{end}
\end{align*}
\end{align*}
\end{align*}
\end{align*}

\begin{align*}
\text{end;} & \quad \text{end.}
\end{align*}

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. If the ring has only ones then $n$ messages are initiated; this happens with probability $2^n$. Otherwise, define a run on the ring to be a segment of zero or more consecutive ones bordered on its left and its right by zeroes. The number of runs on the ring equals the number of zero inputs. Let Max-Run be the number of maximum length runs on the ring; this is the number of messages initiated in the second phase of the algorithm. We terminate the proof by showing that $E(\text{MaxRun}) = O(1)$. In order to do so we need the following auxiliary result, which is proven by an argument similar to that used by Rabin [Ra].

Lemma 3.7: Consider $r$ independent sequences of Bernoulli trials, with probability $p = 0.5$ of success. Let $M_r$ be the number of sequences where the waiting time for the first success is maximum. Then $E(M_r) \leq 2$.

Order the runs on the ring. The last lemma shows that the expected number of maximal length runs among the first $r$ runs on the ring is constant, for any fixed number $r$. We conclude the proof by showing that with probability $> 1 - 1/n$ all the maximal length runs on the ring are among the first $r = n/2 - \sqrt{n \log n}$ runs.

Theorem 3.8: $E(\text{MaxRun}) = O(1)$.

Proof Outline: We may assume that the inputs at processors 1, 2, \ldots, $n$ are generated by a sequence of $n$ Bernoulli trials, with $p = 0.5$. Let $Y$ be the number of maximum length runs in this sequence, ignoring wrap-around. Clearly, $\text{MaxRun} \leq Y+1$. Let $F$ be the number of failures in the sequence. Define the following events:

\begin{align*}
A_1 & := \{ n/2 - \sqrt{n \log n} < F < n/2 + \sqrt{n \log n} \} ; \\
A_2 & := \{ \text{there is a run of length } \geq 0.8 \log n \text{ among the first } n/2 - \sqrt{n \log n} \text{ runs} \} ; \\
A_3 & := \{ \text{there is no run of length } > 0.8 \log n \text{ among the next } 2\sqrt{n \log n} \text{ runs} \} .
\end{align*}

Let $A = A_1 \cup A_2 \cup A_3$. Then, since $Y \leq n$,

$$
E(Y) \leq E(Y \mid A) \text{Prob}[A] + n(1 - \text{Prob}[A]) .
$$

If $A$ occurs then $Y$ is the number of maximum length runs within the first
335

\[ n/2 - \sqrt{n \log n} \text{ runs. Since each run is a recurrent event, it follows that} \]

\[ E(Y \mid A) \text{Prob}[A] \leq E(M_{n/2 - \sqrt{n \log n}}) \leq 2. \]

We conclude the proof by showing that

\[ \text{Prob}[A_i] \geq 1 - 1/n, \text{ for } i = 1, 2, 3. \]

The bound on the probability of \( A_1 \) follows from Chernoff's bounds on the tail of a Bernoulli distribution [Ch, Rag]; the bounds on the probabilities of \( A_2 \) and \( A_3 \) follow from a straightforward analysis. \( \square \)

We have proven

**Theorem 3.9:** For each \( n \) there exist a deterministic synchronous input collection algorithm \( SIC_n \) that has average message complexity \( C_{\text{aver}}(SIC_n) = O(n) \).

The last algorithm can be used to elect a leader in a synchronous labeled ring with \( O(n) \) messages on the average, using only comparisons. Assume the ring is oriented. Then each processor computes a bit by comparing its id to the id of its left neighbor. The last algorithm is then run, using these bits as inputs. In addition to collecting the "input" bits, one also collects the processor id's. The processor with the largest id is then elected as leader. This algorithm can be extended to nonoriented rings.

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. We obtain

**Theorem 3.10:** For each \( n \) there exist an errorless probabilistic synchronous input collection algorithm \( PSIC_n \) that uses a unique random bit per processor, and has expected message complexity on any input \( C_{\text{max}}(PSIC_n) = O(n) \).

### 4. LOWER BOUNDS

#### 4.1. 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.

**Definition:** A function \( f : \Sigma^* \rightarrow \Sigma \) is nonlocal if there exist a constant \( 0 < c < 1 \) such that for any string \( s \) of length \( |s| \leq cn \) there exist two strings \( t_1 \) and \( t_2 \) such that \( |s \cdot t_1| = |s \cdot t_2| = n \) and \( f(s \cdot t_1) \neq f(s \cdot t_2) \).

The function \( f \) is nonlocal if its value can not be determined from the value of a small prefix. The XOR function is nonlocal. More generally, we say that a computation problem is nonlocal if there is a constant \( 0 < c < 0.5 \) such that on any initial input, the output at a processor can not be determined from the value of its \( cn \)-neighborhood. We have

**Lemma 4.2:** The orientation problem is nonlocal.

The proof of the following lemma is immediate from the definition of the synchronizing adversary and Le. 4.1.

**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 $A_n$ be an asynchronous algorithm that solves a nonlocal problem on rings of size $n$. Consider a computation of $A_n$ under the synchronizing scheduler. Then a message is sent by some processor at each cycle $i$, for $i = 1, \ldots, cn$.

Let $A_n$ be an algorithm for rings of size $n$. Let $S_k(A_n)$ be the set of $k$-neighborhoods that cause a message to be generated by the processor with that neighborhood at the $k$-th cycle, when the algorithm $A_n$ is executed with the synchronizing scheduler. Our proof uses a counting argument similar to that used by Bodleander [B] to show that the sets $S_k(A_n)$ are large. Since rings are bidirectional, a more delicate counting argument is needed. Let $N(s)$ be the set of strings that appear cyclically in string $s$ (i.e. equal to a prefix of some cyclic shift of $s$). We have

**Lemma 4.5:** Let $A_n$ be an asynchronous deterministic algorithm that solves a nonlocal problem on rings of size $n$. Let $s$ be the configuration of $k$ successive processors on a ring, where $k \mid n$ and $k < cn$; let $r$ be an integer such that $2r + 1 < k$. Then $N(s) \cap S_r(A_n) \neq \emptyset$.

**Proof:** Denote $l = n / k$ and look at the configuration $C = s^l$. By Corollary 4.4 a message is sent by some processor at each cycle $r$, $r = 1, \ldots, k/2$ on the computation of $A$ on $C$ under the synchronizing adversary. The $r$-neighborhood of this processor is the required string. 

**Corollary 4.6:** Let $A_n$, $r$ and $k$ be as in the previous lemma. Then

$$| S_r(A_n) | \geq 2^{2r+1} / k.$$  

**Theorem 4.7:** Let $A_n$ be an asynchronous deterministic algorithm that solves a nonlocal problem on rings of size $n$. Let $d_1 < d_2 < \cdots < d_r$ be the sequence of divisors of $n$, ordered in increasing order. Then

$$C_{\text{aver}}(A_n) \geq 0.5n \sum_{d_i \leq cn} (1 - d_{i-1} / d_i).$$  

**Proof:** Let $s$ be a random input of length $n$. Let $t$ be the $r$-neighborhood of a fixed processor in $s$, where $d_{i-1} < 2r + 1 < d_i \leq cn$. According to Cor. 4.6,

$$\text{Prob}\{t \in S_r(A_n)\} \geq 1 / d_i.$$  

Thus, the expected number of messages sent at cycle $r$ on input $s$ is $\geq n / d_i$, for $r \leq d_i \leq cn$. The total expected number of messages is obtained by summing:

$$C_{\text{aver}}(A_n) \geq \sum_{d_i \leq cn} \sum_{d_{i-1} < k \leq d_i, \text{ odd}} n / d_i \geq 0.5n \sum_{d_i \leq cn} (1 - d_{i-1} / d_i).$$

**Corollary 4.8:** Let $n = 2^k$, then for any asynchronous deterministic algorithm $A_n$ solving a nonlocal problem on rings of size $n$, $C_{\text{aver}}(A_n) = \Omega(n \log n)$.

Since XOR and orientation are nonlocal it follows:

**Corollary 4.9:** Let $n = 2^k$, then for any asynchronous deterministic algorithm $A_n$ computing XOR or orientation on rings of size $n$, $C_{\text{aver}}(A_n) = \Omega(n \log n)$.

4.2. Probabilistic Algorithms

Let $A_n$ be an errorless probabilistic algorithm with input alphabet $\Sigma$. Assume there is a fixed upper bound on the number of messages sent by $A_n$ (the bound may depend on $n$). We can replace an execution of $A_n$ by a computation whereby each processor
first chooses independently a random binary string of fixed length \( q \), next runs a deterministic algorithm \( A_n^d \) (with input set \( \Sigma \times \{0,1\}^q \)). The resulting algorithm is errorless and the average complexity of \( A_n^d \) can be made arbitrarily close to the average complexity of \( A_n \). If \( A_n \) solves a nonlocal problem, then \( A_n^d \) solves, too, a nonlocal problem. Thus, Cor. 4.9 implies the following result.

**Theorem 4.10:** Let \( A_n \) be an errorless probabilistic asynchronous algorithm that solves a nonlocal problem on an anonymous ring of size \( n = 2^k \). Then \( \Omega_{\text{aver}}(A_n) = \Omega(n \log n) \).

In particular, any errorless probabilistic algorithm that computes XOR or orients a ring has average expected complexity \( \Omega(n \log n) \). This implies an \( \Omega(n \log n) \) lower bound for the expected number of messages sent on the worst input, for any nonlocal problem.

The last \( \Omega(n \log n) \) lower bound on average complexity does not apply to the AND function. Indeed, we have shown in Th. 3.4 that AND can be computed with a linear average number of messages. Nevertheless, we can prove an \( \Omega(n \log n) \) lower bound on the expected number of messages sent on the worst input.

**Theorem 4.11:** Let \( A_n \) be an errorless probabilistic asynchronous algorithm that computes the AND of \( n \) inputs. Then the expected number of messages transferred in the computation with input \( 1, \ldots, 1 \) is \( \Omega(n \log n) \).

**Proof:** Let \( A_n^d \) be the associated deterministic algorithm, and consider the computations of this algorithm on inputs of the form \( 1s_1, \ldots, 1s_n \) under the synchronizing scheduler. The average number of messages sent on these inputs by \( A_n^d \) is equal to the expected number of messages sent by \( A_n \) on input \( 1, \ldots, 1 \). Since \( \text{AND}(1, \ldots, 1) \neq \text{AND}(1, \ldots, 1, 0) \), \( A_n \) must run for at least \( n/2 \) cycles on the input \( 1, \ldots, 1 \). Thus, a computation of \( A_n^d \) on any input of the form \( 1s_1, \ldots, 1s_n \) runs for at least \( n/2 \) cycles. The lower bound follows, using the same arguments as in Le. 4.5, Cor. 4.6, and Th. 4.7.

The complexity of AND can be reduced to \( O(n) \) if a fixed positive probability for error is allowed. The simple algorithm given in Th. 3.6 either gives the right answer, or deadlocks. If we insist that the algorithm never deadlocks, so that the only failure mode is a wrong answer, then the situation is different; almost nothing is saved by allowing errors. We have

**Theorem 4.12:** Let \( A_n \) be a probabilistic asynchronous algorithm that computes the AND of \( n = 2^t \) inputs with error \( p \), and never deadlocks. Then

\[
\Omega_{\text{aver}} = \Omega(n (\log n - \log\log(1/(1-p)))).
\]

**Proof:** See [AS].

The last bound implies that a probabilistic asynchronous algorithm that computes AND and never deadlocks requires \( \Omega(n \log n) \) messages, as long as the probability of success is significantly larger than \( 2^{-n} \). Using this result, we can show that Th. 4.11 is valid even for unbounded computations; the same holds true for XOR or orientation. Interestingly, it is possible to compute AND with \( O(n) \) messages on a ring of odd length \( n \), with a probability of success \( n \cdot 2^{-n} \), and no deadlock; thus, the last result is optimal.

5. **CONCLUDING REMARKS**

We have shown in Th. 3.4 that whenever there exists a string of length less than \( \log\log n - \log\log\log n \) that determines the value of a function \( f \), then \( f \) can be computed in less than \( n \log n \) 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 input collection algorithm uses messages containing up to \( n \) bits. It is easy to modify the algorithm so that no more than \( O(n^2) \) bits are transferred on the average. Simple information transfer arguments show that this is optimal. The number of bit transfers can be reduced for XOR and orientation to \( O(n \log^2 n) \). 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.

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 [DG]). This scheduler keeps the computation as symmetric as possible. Here, as in [ASW], the lower bounds reflect the cost of breaking symmetry.

ACKNOWLEDGEMENTS

We would like to thank Michael Benor, Rafi Hassin and Prabhakar Raghavan for their useful suggestions.

REFERENCES:


