GAP THEOREMS FOR DISTRIBUTED COMPUTATION

by

S. Moran* and M.K. Warmuth

Technical Report #449
March 1987

* Département of Computer Science, Technion-\textsc{tit}, Haifa, Israel

** Department of Computer and Information Sciences, University of California, Santa Cruz, CA 95064
Gap Theorems
for Distributed Computation
(revised version)

Shlomo Moran *
Department of Computer Science, the Technion,
Haifa 32000, Israel.

Manfred K. Warmuth **
Department of Computer and Information Sciences,
University of California,
Santa Cruz, CA 95064.

* Part of this work was done while this author was at IBM Thomas J.
Watson Research Center, Yorktown Heights, NY 10598.
** This research was supported by Faculty Research funds granted by the University of California, Santa Cruz.
KEY WORDS:

Distributed algorithms, networks, asynchronous, message complexity, bit complexity, ring of processors, lower bounds, gap theorem.

ABSTRACT:

Consider a ring of \( n \) anonymous processors, i.e., the processors have no id’s. Each processor receives an input string, and the ring is to compute a function of the circular input configuration in the asynchronous bidirectional model of computation. The complexity of an algorithm is the number of bits or the number of messages sent in the worst case. The complexity of a function is the lowest complexity of any algorithm that computes that function. If the function value is constant for all input configurations, the processors do not need to send any messages (complexity: zero). On the other hand, we prove that any non-constant function has bit complexity \( \Omega(n \log n) \) for anonymous rings. There are non-constant functions that reach the upper end of the gap, i.e., we exhibit a non-constant function of bit complexity \( O(n \log n) \). The same gap for the bit complexity of non-constant functions remains even if the processors have distinct id’s, provided that the id’s are taken from a large enough domain.

For the case of using the number of messages sent rather than the number of bits as the complexity measure, we present a non-constant function that can be computed with \( O(n \log^+ n) \) messages on an anonymous ring.
0. INTRODUCTION

There has been an extensive amount of research on studying computation on a ring of \( n \) asynchronous processors. The ring topology is in a sense the simplest distributed network that produces many typical phenomena of distributed computation. In this model the processors may communicate by sending messages along the links of the ring, which are either unidirectional or bidirectional. All the messages sent reach their targets after a finite, but unpredictable and unbounded delay. Numerous algorithms [ASW87, DKR82, P82] have been found for the asynchronous ring. All these algorithms require the transmission of \( \Omega(n \log n) \) bits. This is not surprising in view of the results of this paper. We establish a gap theorem for asynchronous distributed computation on the ring which says that either the function computed is constant and no messages need to be sent, or, in case of an arbitrary non-constant function, \( \Omega(n \log n) \) bits are required. In the proof we cannot rely on the particular properties of a function such as extrema finding [B80, PKR84]. Rather than that, we prove a general lower bound of \( \Omega(n \log n) \) for arbitrary non-constant functions.

In our model there is no leader among the \( n \) processors. All processors run the same program which may depend on the ring size. We first treat the case where the ring is unidirectional and the processors have no id’s (the anonymous model of [ASW87]). Then we show that the lower bound holds also for bidirectional rings, and for rings of processors with distinct id’s, provided the set of possible id’s is sufficiently large. Note that in the anonymous ring without a leader it is necessary that the processors “know” the ring size. Otherwise the processors cannot determine when to terminate [ASW81].

Let us contrast the anonymous ring with the model consisting of a ring of asynchronous processors with a leader. If we assume unidirectional communication then any non-constant function must require \( \Omega(n) \) bits. But in the bidirectional case, there are simple non-constant functions for any bit complexity \( O(c(n)) \). For example, assume that the inputs are bits and \( c(n) = n^2 \). The input is denoted as an \( n \)-bit word \( \omega \), the \( i \)-th bit \( \omega_i \) of \( \omega \) being the input of the \( i \)-th processor. It is not hard to verify that the following function is a non-constant function of bit complexity \( \Theta(c(n)) \): \( f(\omega) = 1 \) iff \( \omega \) contains a palindrome of \( 2 \sqrt{c(n)} + 1 \) bits centered at the leader. Thus there is no gap for rings with a leader. Our gap theorem for anonymous rings clearly quantifies the price one has to pay for having no distinguished processor. We show that the gap theorem holds even if the processors have distinct id’s, provided that the set of id’s from which the distinct id’s are chosen is large enough.

The same gap of \( \Omega(n \log n) \) for anonymous rings does not hold if one counts messages (of arbitrary length) instead of bits. In [ASW85] a non-constant function was presented that is computable in \( O(n) \) messages on an anonymous ring. This function is only defined for rings of odd size \( n \). It is easy to find similar functions for rings whose sizes are not divisible by some fixed integer. In the case where the smallest non-divisor of the ring size is large, the ring seems to be more “symmetric”, and it is hard to find non-constant functions of low message complexity. We exhibit a non-constant function for arbitrary ring size that requires \( O(n \log^+ n) \) mes-
A different sort of "gap" has been found recently by Mansour and Zaks [MZ85]: consider the language of input configurations recognized by an asynchronous ring algorithm with a leader; this language has bit complexity $O(n)$ iff it is regular; furthermore every non-regular language requires $\Omega(n \log n)$ bits. There is a crucial difference between the model of [MZ85] and our model. They assume that the processors (including the leader) do not know the ring size. As mentioned above, there are no gaps if there is a leader and the processors do know the size of the ring (which is a necessary condition [ASW87] for the anonymous ring without a leader as in our model). Without the knowledge of the ring size the processors can only compute regular languages on a ring with a leader, and non-regular languages require $\Omega(n \log n)$ bits. Note that this is exactly the number of bits required for computing the ring size and intuitively this fact is responsible for the gap shown in [MZ85]. The results of [MZ85] are analogous to the classical results for one-tape Turing machines [T64, H68].

Our lower bound proofs rely heavily on the asynchronous nature of the computation. We use the fact that the result of a computation must be independent on the specific delays times of the links, and impose certain delays on the links that assure that $\Omega(n \log n)$ bits are sent. Note that in the synchronous anonymous ring the Boolean AND can be computed with $O(n)$ bits [ASW87], and $\Omega(n)$ is also a trivial lower bound for an arbitrary non-constant function.

$\Omega(n \log n)$ is the bit complexity inherent to the ring, since any non-constant function must have this bit complexity when computed on the ring. Our research opens many challenging problems concerning gap theorems in distributed computation. Given a network of anonymous processors, define the distributed bit (message) complexity of the network to be the smallest bit (message) complexity of a non-constant function when computed on that network. Intuitively, this complexity measures the minimum effort needed to coordinate the processors of the network in any sensible way. This coordination should be more difficult if the network is highly symmetric. What parameters of the network correspond to this complexity? How does this complexity depend on the connectivity, diameter, etc? Our results show that the distributed bit complexity of a ring of $n$ processors is $\Theta(n \log n)$.

In the next section we summarize the model of computation. In Second 2 we first prove the $\Omega(n \log n)$ lower bound on the bit complexity for unidirectional rings with no id's, and then generalize the result to bidirectional rings and to rings with id's. In the last section we show that for each ring size $n$, there are non-constant functions that can be computed on a ring of $n$ anonymous processors by algorithms of $O(n \log n)$ message complexity and $O(n \log n)$ bit complexity, respectively. The functions are defined by interleaving de Bruijn sequences of various lengths.

\footnote{The function $\log_2 n$ grows very slowly (<6 for $n \leq 2^{65536}$). It is the number of iterations of the function $\log_2$ required to get the value $n$ down to <1.}
The Gap Theorem with distinct id’s assumes that the id’s are chosen from a rather large set. This raises the open problem of whether $\Omega(n \log n)$ bits are required for any non-constant function even if the set of id’s is small (for example $2n$). Furthermore, the Gap Theorems should be generalized to probabilistic models.

Note that in our definition functions do not depend on the distinct id’s. Allowing such a dependence gives rise to simple non-constant functions that can be computed with $O(n)$ bits: If there are only $n$ possible id’s then the processor having the smallest id can be considered as a leader, and we have discussed this case above; otherwise, the function which equals one iff the distinct id’s of the given ring are the $n$ smallest id’s can be computed in $O(n)$ bit complexity.

1. THE MODEL

Our computational model is a ring $R$ of $n$ processors, $P_1P_2 \cdots P_n$. The processors are anonymous, i.e., they are not given their index on the ring and they run the identical program which may depend on the ring size. Consecutive processor (and $P_nP_1$) are connected via communication links. In the bidirectional ring connected processors can send messages towards each other via these links. In the unidirectional ring messages can only travel in one direction around the ring. The messages sent along a certain direction of a link are assumed to arrive in order which they have been sent. The communication is asynchronous, meaning that messages arrive with an unpredictable but finite delay time.

In our model we assume that the input of each processor is a letter of arbitrary alphabet $I$, i.e., the functions have the domain $I^n$. We shall assume that $I$ contains the letter 0. Functions computed on a ring without a leader must be invariant under circular shifts [ASW87]. The essential property of the function that we want to capture is that it is non-constant. Thus we may assume that the non-constant function is a characteristic function of a non-trivial subset $S$ of $I^n$. Note that non-constant functions require that the size of $I$ is at least two. We shall also assume that upon termination of the algorithm, every processor in the ring is either in an “accepting” state (output 1) or in a “rejecting” state (output 0).

2. THE GENERAL LOWER BOUND ON THE BIT COMPLEXITY

The main theorem proved in this section is the following:

Theorem 1: For a unidirectional ring of $n$ anonymous processors, the bit complexity of any non-constant function is $\Omega(n \log n)$.

The proof of the theorem will follow from some lemmas, given below. We will give lower bounds on the worst case complexity of an arbitrary algorithm $AL$ that computes the characteristic function of any non-trivial set $S$ of $I^n$. The first lemma is similar to Theorem 5.1 of [ASW87]. We repeat its proof here, since it uses a technique that is applied in this paper as well. Since the function value must be the same for all possible delay times of the asynchronous computation, we may choose particular delay times for the proofs: all processors start at time zero, internal computation at a processor takes no time and links are either blocked (very
large delay) or are synchronized (it takes exactly one time unit to traverse the link). For the lemma below we assume that all links are synchronized. Intuitively, this keeps the computation symmetric and causes the most messages to be sent.

Lemma 1: Let $AL$ be an algorithm for a bidirectional ring of $n$ processors. If $AL$ rejects $0^n$ and accepts $0^t \cdot t$ for some $t$, then $AL$ requires at least $n \lfloor z/2 \rfloor$ messages on input $0^n$.

Proof: Consider the execution of $AL$ with input $0^n$. The input is completely symmetric. All processors run the same algorithm and thus are in the same state of the algorithm w.r.t each other. At least one message is sent by each processor at each time step until some time step $T$ at which no message is sent. From then on the processor cannot change due to new messages. Thus the processors terminate at time $T$ after sending at least $n(T-1)$ messages altogether. Now consider a second execution with input $0^t \cdot t$. If $T \leq \lfloor z/2 \rfloor$ then the processor $p_{\lfloor z/2 \rfloor}$ is in the same state of the algorithm at time step $T$ in both executions. Thus in both cases $p_{\lfloor z/2 \rfloor}$ terminates with the same result, which is a contradiction. We conclude that at least $n \lfloor z/2 \rfloor$ messages are sent in the execution with $0^n$ as input. □

Without loss of generality, assume that $0^n$ is not in $S$. Otherwise, replace the function and $S$ by its complement. Throughout this section $0 \in I^n$ is a fixed word in $S$, i.e., the function values at $0^n$ and $\omega$ are different. Consider an execution of the algorithm $AL$ on ring $R$ with $\omega$ as input and all links synchronized. Suppose all processors terminate before time $t$. We use a notion of "history" of a processor. The sequence of messages sent by an anonymous processor in an asynchronous computation will be uniquely determined by its input and history. In Theorem 1 we first deal with the unidirectional case (all messages are received from the left). For $0 \leq s \leq t$ and for $1 \leq i \leq n$ we define $h_i(s)$, the history of processor $i$ after $s$ time units, to be the string $h_i(s) = m_i(1)L \cdots L m_i(k)$, where $L$ is a separating symbol not in $I$, and $m_i(1), \ldots, m_i(k)$ are the messages (words over the alphabet $\{0,1\}$) received by $p_i$ until (and including) time $s$, in this order. Note that $k$ might be smaller than $s$. $H_i = h_i(t)$ is the total history, or simply, the history of $p_i$ in this computation. Note that the total length of $H_i$ is less than twice the number of bits received by $p_i$. Thus, a lower bound on the bit complexity of $AL$ is implied by a lower bound on the sum of the lengths of the histories of the processors in a certain computation of $AL$. The lower bound on the sum of the lengths will follow from the fact that during a certain execution of $AL$ the number of distinct histories of the processors is $\Omega(n)$, and therefore the lemma below implies that $\Omega(n \log n)$ bound.

Lemma 2: Let $H_1, \ldots, H_k$ be $k$ distinct words over an alphabet of size $r > 1$. Then $|H_1| + |H_2| + \cdots + |H_k| > (k/2) \log_r (k/2)$.

Proof: Represent the $H_i$ with an $r$-ary tree, s.t. each $H_i$ corresponds to a path from the root to an internal node or a leaf of the tree. In the tree each leaf is responsible for some $H_i$, but some internal nodes are not responsible for any $H_i$. Assume the overall length of the $H_i$ is minimized. Then in the corresponding tree each leaf and each internal node is responsible for some $H_i$. Furthermore, all nodes except possible the nodes of maximum level have degree $r$, and at least half of the nodes are leaves. The lemma is implied by the fact that the average height of
the leaves in an r-ary tree with v leaves is at least \( \log_r v \). □

Outline of the proof of Theorem 1: An execution of AL is constructed for which either the one Lemma or the other Lemma implies the lower bound of \( \Omega(n \log n) \) bits. In the first case a processor accepts an input that contains \( \log n \) consecutive zeros. Thus Lemma 1 implies \( \Omega(n \log n) \) messages. In the second case there will be an execution with more than \( n - \log n \) processors with distinct histories and Lemma 2 gives an \( \Omega(n \log n) \) bits lower bound.

To simplify notations, assume that \( t = kn \) for some integer \( k \), and let \( C \) be a line of \( t \) processors, denoted by \( p_{1,1}, p_{2,1}, \ldots, p_{n,k} \). Informally, \( C \) consists of \( k \) copies of the ring \( R \) of \( n \) processors that were cut at the link \( p_n = p_1 \) and then concatenated to one line of \( kn \) processors. Thus, processor \( p_{i,j} \) in \( C \) corresponds to the processor \( p_i \) in the \( j \) th copy of \( R \). We make \( C \) a ring by connecting \( p_{n,k} \) with \( p_{1,1} \) by a link which is blocked. Note that even though every processor in \( C \) acts as if it is on a ring, the block on the link \( p_{n,k} = p_{1,1} \) makes the global behavior of \( C \) to be that of a line of processors.

Let \( \omega_k \) be the input to \( C \), where \( p_{i,j} \) receives the letter \( \omega_i \) as an input, and consider the execution of AL on \( C \) in which all links are synchronized except for the block on the link \( p_{n,k} = p_{1,1} \). For \( 0 \leq r \leq k \), the histories \( h_{i,j}(s) \) and \( H_{i,j} \) of the processor \( p_{i,j} \) in \( C \) (with input \( \omega_k \)) are defined similarly to the histories of the processors in \( R \) (with input \( \omega \)) given above. Remember that all processors of \( R \) terminate at time \( t-1 \) or before. Using an argument similar to the "shifting scenario" argument of [FLM85] we show that \( p_n \) and \( p_{n,k} \) act alike.

Lemma 3: Processor \( p_{n,k} \) accepts.

Proof: Remember that \( C \) consists of \( k \) identical copies of \( R \). Assume for a moment that there is no block on the link \( p_{n,k} = p_{1,1} \), i.e., that all links are synchronized. It is easy to see that in that case \( h_{i,j}(s) = h_{i}(s) \), for all \( i, j \) and for \( 0 \leq r \leq k \). If we now restore the block on \( p_{n,k} = p_{1,1} \), then by time \( s \), the block can only effect the \( s \) leftmost processors. Thus at time \( t-1 \) processor \( p_{n,k} \) has exactly the same history as \( p_n \) has at time \( t-1 \) and \( p_{n,k} \) accepts because \( p_n \) does so. □

Observe that there may be many processors in \( C \) with identical (total) histories. We define a subsequence \( \overline{C} \) of \( C \) s.t. all of its processors have distinct histories at the end of the computation. First we use \( C \) to construct a directed graph \( G \), and then construct \( \overline{C} \) from \( G \).

The vertices of \( G \) are the processors of \( C \), and there is a directed edge from \( p \) to \( q \) if \( q \) is the rightmost processor having the same history as the processor to the right of \( p \). It is easy to see that there is exactly one edge leaving every processor except the last processor, \( p_{n,k} \), and that \( G \) contains no cycles. Thus, \( G \) is a directed tree rooted at \( p_{n,k} \). \( \overline{C} \) is now the sequence of processors on the unique path that starts at \( p_{1,1} \) and ends at \( p_{n,k} \).

Lemma 4: No two processors of \( \overline{C} \) have the same history in the execution on \( C \).

Proof: The first processor in \( \overline{C}, p_{1,1} \), is the only processor that receives no messages during the computation described above, due to the block on the link entering it. For the other processors the lemma follows from the fact that for each history \( H \) there is at most one rightmost
Let the sequence of processors \( C \) defined above be \( C = (p_{i_1,j_1}, \ldots, p_{i_m,j_m}) \), and let \( \tau \) be the input word \( \omega_{i_1} \cdots \omega_{i_m} \) of length \( m \). Note that \( p_{i,j} \) gets \( \omega_i = \tau_i \) as an input. We now run \( AL \) on \( C \) with all links synchronized except for the link \( p_{i_m,j_m} - p_{i_1,j_1} \), which is blocked.

**Lemma 5:** In the execution of \( AL \) on \( C \) with input \( \tau \), the history of processor \( p_{i_j,j} \) \((1 \leq l \leq m)\) of \( C \) is the same as the history of \( p_{i_j,j} \) in the execution of \( AL \) on \( C \) with input \( \omega^\tau \). In particular, processor \( p_{i_m,j_m} (= p_n, k) \) of \( C \) accepts.

**Proof:** This follows from a simple induction of \( I \) using the way \( C \) is constructed from \( C \) and from the fact that the input and the history of a processor determines the messages sent by the processor.

**Corollary 1:** For any \( 1 \leq l \leq m \), the number of bits received by \( L \) distinct processors of \( C \) in the execution described above is at least \((1/4) \log_3 (3/2)\).

**Proof:** Let \( L \) be a set of \( l \) processors in \( C \), and let \( p \in L \). By Lemma 5, the history of \( p \) in the computation of \( C \) is the same as its history in the computation of \( C \). This implies, by Lemma 4, that no two processors in \( L \) have the same history. Thus, by Lemma 2, the sum of the lengths of the histories of the processors in \( L \) is at least \((l/2) \log_3 (l/2)\). The lemma follows by the observation made earlier, that this sum is less than twice the number of bits received by these processors.

**Proof of Theorem 1:** Let \( \tau^* \) be the first \( n \) letters of \( \tau \), where \( * \) denotes the Kleene star. Let \( m' = \min((m, n)) \) (recall that \( |\tau| = m \)). Consider a computation of \( AL \) on \( \tau^* \) in which the first \( m' \) processors of \( C \) have exactly the same history as the corresponding first \( m' \) processors in the computation of \( C \) on \( \tau \) described above, and no message sent by the remaining processors is ever received. We distinguish two cases depending on the length \( m \), of \( C \).

**Case** \( m \leq n - \log n \): By Lemma 5, \( \tau^* \) will be accepted by at least one processor in \( R \). Since \( \tau^* \) ends with \( \log n \) zeros, Lemma 1 guarantees that \( \Omega(n \log n) \) messages are required for the input \( 0^n \) and the theorem holds.

**Case** \( m > n - \log n \): By Corollary 1, the total number of bits received by the first \( m' \) processors is at least \((m'/4) \log_3 (m'/2)\) which at least \( \Omega(n \log n) \), and this completes the proof of Theorem 1.

The proof of the gap theorem for bidirectional rings follows the same general outline, but is trickier.

**Theorem 1':** For a bidirectional ring of \( n \) anonymous processors, the bit complexity of any non-constant function is \( \Omega(n \log n) \).

**Proof:** For simplicity, we shall assume that the ring \( R \) is oriented (i.e., all the processors in it agree on the same "right" and "left" directions). Let \( AL \) be an algorithm for a bidirectional ring \( R \) of size \( n \), and assume that it recognizes a non-constant set \( S \) over \( I^n \). Assume further
that $0^n$ is not in $S$. Let $w$ be a word in $S$, and again consider a "synchronized" computation of $AL$ on $w$. The history of a processor $p_i$ at time $s$ in such a computation is a string $h_i(s) = d_i(1)m_i(1) \ldots d_i(k)m_i(k)$, where $d_i(j)$ is either $R$ (for right) or $L$ (for left), and the $m_i(j)$'s are the distinct messages received by $p_i$, up to time $s$, in this order; $m_j$ is received from direction $d_i(j)$. Note that the length of $H_i$ is at most two times larger than the number of bits received by $p_i$.

Assume that $AL$ accepts $w$ in less than $t$ time units, where $t = nk$. Define a line $D$ of $2t = 2nk$ processors as follows: Let $C$ be a line of $nk$ processors as defined before, and let $C'$ be the line obtained by replacing each $p_{i,j}$ in $C$ by $p'_{i,j}$. $D$ is constructed by connecting the last processor $p_{n,k}$ of $C$ with the first processor $p'_{1,1}$ of $C'$, and by connecting $p'_{n,k}$ to $p_{1,1}$.

We consider a particular execution of $AL$ on $D$ with input $w^{2k}$. Again internal computation of a processor takes no time and a message requires exactly one unit to traverse a link. A processor is blocked at time $s$ if it receives no messages at time $s$ or later. In the chosen execution $p_{1,1}$ and $p'_{n,k}$ are blocked at time one, and in general at time $s$ $(1 \leq s \leq t)$, the $s$ leftmost and the $s$ rightmost processors of $D$ are blocked. Note that no message sent on the link $p'_{n,k} \rightarrow p_{1,1}$ is ever received and thus $D$ acts as line of processors. The following stronger version of Lemma 3 holds for the above execution on $D$. In this version, $h'_{i,j}$ denotes the history of processor $p'_{i,j}$.

Lemma 3': Let $p_{i,j}$ [$p'_{i,j}$] be the $s$th leftmost [rightmost] processor in $D$ $(1 \leq s \leq t)$. Then $h_{i,j}(t) = h_j(s-1)$ [$h'_{i,j}(t) = h'_j(s-1)$]. That is, at the end of the above execution on $D$, the history of the $s$th leftmost [rightmost] processor in $D$ is equal to that of the corresponding processor in $R$ after $s-1$ time units. In particular, $p_{n,k}$ and $p'_{1,1}$ accept.

Since every $n$ consecutive processors in $D$ corresponds to the $n$ processors of $R$, Lemma 3' implies:

Corollary 3: Let $R'$ be a set of $n$ consecutive processors in $D$ corresponding to a copy of $R$. Then the sum of the lengths of their histories is not larger than the sum of the lengths of the histories of the processors in $R$.

With the sequence $D$ described above we associate a digraph similar to the one associated with $C$: The vertices are the processors in $D$; recall that $D = C \cdot C'$; there is an edge from each processor $p$ in $C$ to the rightmost processor in $C$ which has the same history as the right neighbor of $p$, and there is an edge from each processor $p'$ in $C'$ to the leftmost processor in $C'$ which has the same history as the left neighbor of $p'$. We also add an extra edge from the leftmost processor in $C'$, $p'_{1,1}$, to the rightmost processor of $C$, $p_{n,k}$. It is easily observed that this graph is a directed graph rooted at $p_{n,k}$. $\overline{C} \cdot \overline{C'}$ (or $C \cdot C'$) are then taken to be the unique paths from $p_{1,1}$ ($p'_{n,k}$) to $p_{n,k}$. $\overline{D}$ is the concatenation $\overline{C} \cdot \overline{C'}$.

The construction of $\overline{C}$ and $\overline{C'}$ guarantees that no two processors in $C$ or in $\overline{C'}$ have the same history in the execution on $\overline{D}$ described above, hence at most two processors in $\overline{D}$ have the same history. The proof that there is a computation on $\overline{D}$ in which every processor has the same history it has in the computation on $D$ is similar to the proof of Lemma 5 for the.
unidirectional case. Using the above, we bound the number of bits received by any \( l \) distinct processors in \( C \) or in \( C' \) by \((l/4)\log_4(1/2)\).

Let \( m \) be the length of \( D \). In the case where \( m \leq n \) we can pad \( D \) with \( n - m \) processors that receive input zero. As in the proof Theorem 1, the messages sent by these \( n - m \) processors never reach their target. According to Lemma 3' \( p_{n,k} \) accepts. By distinguishing two cases depending on whether \( m \leq n - \log n \) or not, one can complete the proof of Theorem 1 as before, provided that \( m \leq n \).

However if \( m > n \) then one cannot proceed as in the unidirectional case. If we would cut the \( m \) leftmost processors of \( D \), then the last processor does not receive the proper messages from the right, and the proof requires modification. For \( b = 1, \ldots, k \), let \( D_b \) be a linear arrangement of \( 2nb \) processors \((D_k = D)\); observe that \( D_{b+1} \) is obtained by inserting a line of \( 2n \) processors between the left and the right halves of \( D_b \).

For each \( D_b \) consider a computation on input \( \omega^{2b} \), in which the processors are blocked similarly to the way they were blocked in the computation on \( D \) described above. Use this computation to define \( D_b \) for \( b = 1, \ldots, k \), and let \( m_b \) be the number of processors in \( D_b \). Then one can verify that \( m_{b+1} - m_b \) is bounded by the number of distinct histories in the set of the \( 2n \) consecutive processors of \( D_{b+1} - D_b \). Also, we have that:

(i) \( m_b > n \) (by the assumption on \( m = m_k \)), and
(ii) \( m_1 \leq m_2 \leq \cdots \leq m_k \) (by inclusion).

This implies that either there exists a \( b \) such that \( m_{b+1} - m_b \geq \frac{n}{2} \) (with \( m_0 = 0 \)), or there is a \( b \) such that \( \frac{n}{2} \leq m_b \leq n \).

In the former case we have that there are \( 2n \) consecutive processors in \( D_{b+1} \) which have at least \( \frac{n}{2} \) distinct histories. This implies that there are \( n \) consecutive processors in \( D_{b+1} \) which have at least \( \frac{n}{4} \) distinct histories and the lower bound follows from by Lemma 2 and Corollary 3. In the second case, the proof follows the idea of the proof of Theorem 1: let \( \tau \) be the word composed of the input letters to the processors in \( D_b \), in the same order. Then \( n/2 \leq m' = |\tau| \leq n \). By considering a computation on input \( \tau' = \tau \omega^{n-m'} \); as defined in the proof of Theorem 1, we obtain a computation which has at least \( n/2 \) distinct histories, and the result follows. This completes the proof of Theorem 1.'
any non-constant function is $\Omega(n \log n)$, provided that the set of id's that can be assigned to the processors is large enough.

Proof: Assume for contradiction that there is an algorithm $AL$ of bit complexity $o(n \log n)$ computing a non-constant function and let $Z$ be the set of possible id's, with $|Z| \geq n 2^n$. Let $H_z$ be the set of possible extended histories that can occur at a processor with id $z \in Z$ when $AL$ is executed. Since the length of the history of a processor is at most three times the number of bits received or sent by that processor, the assumption on the bit complexity of $AL$ implies that the set of all possible extended histories $H = \bigcup_{z \in Z} H_z$ is of cardinality $o \left( 2^n \log^{+} n \right) = o \left( n^n \right)$.

Denote $z = _{AL} z'$ if $H_z = H_{z'}$. (Note that if $z = _{AL} z'$ then the algorithm on input $z$ is identical to the algorithm on input $z'$.) The relation $=_{AL}$ partitions $Z$ into at most $2 |H| = o \left( 2^n \right) = o \left( \frac{|Z|}{n} \right)$ equivalence classes. Thus, there must be $n$ distinct id's in $Z$ which belong to the same equivalence class. By restricting the id's to this equivalence class, the proof of the anonymous case applies directly. $\Box$

3. FUNCTIONS COMPUTABLE WITH A SMALL NUMBER OF MESSAGES

In [ASW85] a non-constant function was presented that is computable in $O(n)$ messages. This function is the characteristic function for strings of the pattern $0(01)^*$, i.e., the function is one if the input string is of this pattern or a cyclical shift thereof, and zero otherwise. Unfortunately, the pattern requires that the ring size $n$ is odd. Similar patterns can be found for string lengths whose smallest non-divisor is a fixed constant. However, we were not able to find non-constant functions for arbitrary ring size that have linear message complexity. In this section we exhibit a function with "almost linear" message complexity, i.e., $O(n \log^{+} n)$, that is defined for arbitrary ring size. As a by-product of the construction, we introduce a set that can be recognized with $O(n \log n)$ bit complexity, showing that the $O(n \log n)$ bounds on the bit complexity given in the previous section are tight. All our algorithms work on the unidirectional ring. It is easy to derive symmetric bidirectional versions of them s.t. the message complexity increases by only a constant factor.

To get an easy introduction we will first repeat the algorithm of [ASW85] used to recognize $0(01)^*$ (called Algorithm $X$). Algorithm $A$, the second algorithm described, is a parameterized version of Algorithm $X$. It follows the same outline as $X$ but is used to recognize more complicated patterns. In particular, $A$ is used to recognize a pattern in $O(n \log n)$ bits. Finally, $A$ is used iteratively in Algorithm $B$ for recognizing a pattern in $O(n \log^{+} n)$ messages.

For the definition of $\log^{+} n$, let $g_0 = 1$ and $g_{i+1} = 2^{g_i}$; $\log^{+} n$ is the minimum $i$ s.t. $g_i > n$. Note that $g_{\log^{+} n - 1} > \log n$, and that $g_{\log^{+} n - 1} > \log n$, and that $\log^{+} n \leq n \preceq \log^{+} n + 1$.

If $f(n) = o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$. 

Technion - Computer Science Department - Technical Report CS0449 - 1987
Algorithm $X(n)$:
(*) $n$ is assumed to be odd*)

For all $n$ processors in parallel do

X1. Send your bit to the right neighbor.
X2. If both the bit received and your bit is a one then send a zero-message to the right and terminate with output zero.
X3. If you receive a zero-message forward it and terminate with output zero.
X4. If both the bit received and your bit is a zero then initiate an $X$-counter message with count one.
Otherwise forward the first $X$-counter message received after increasing its counter by one.
X5. If you initiated an $X$-counter then wait until you receive an $X$-counter. If its count is $n$, initiate a one-message and terminate with result one.
If you initiated an $X$-counter and receive a counter $<n$, initiate a zero-message and terminate with result zero.
Otherwise, i.e. if you did not initiate an $X$-counter, forward the message you receive and terminate with output zero if it was a zero-message and with output one if it was a one-message.

How does Algorithm $X$ recognize the pattern 0(01)∗? Any input string of odd length contains either a 11 pair or a 00 pair. The substring 11 is not a legal substring of the pattern. Thus the algorithms sends a zero-message around the ring (Step X2) which guarantees that all processors halt with output zero. Possible in parallel, the substrings 00 of the input each initiate an $X$-counter (Step X4). Before concluding that the output is one, the algorithm checks if there is exactly one 00 in the input. The $X$-counters travel around the ring until they run into another 00. Only if an $X$-counter traveled for distance $n$ one can conclude that the 00 is unique. Also since an $X$-counter of count $n$ circled the whole ring it assures that no processor terminated with result zero before the final one-message is initiated in Step X5. Note that each processor sends at most two messages, so that the total number of messages is linear. Counters cost at most $\lfloor \log n \rfloor + 1$ bits, so the total bit complexity is $O(n \log n)$.

Note that Algorithm $X$ does not terminate on inputs of the form (01)* which are of even length. Whenever the ring size $n$ has a small fixed non-divisor, then it is easy to find patterns that can be recognized with algorithms similar to $X$. The results of this section on the complexity of non-constant functions should certainly be independent of the size of the input alphabet. The most general case is alphabet size two. Assume we had an algorithm that worked for arbitrary ring size and fixed alphabet size $p$. We could encode the $i$-th letter by $1^i q_{p+1-i}$ and simulate the algorithm on binary input. However, the ring size is now restricted to multiples of $p+1$. We use algorithms similar to $X$ to separately handle all ring sizes that are not divisible by $p+1$. The combined algorithm works again for all sizes. These observations allow us to use a constant number for symbols for our input patterns.

We next define a pattern which is based on the de Bruijn sequences [B46] that can be
recognized in $O(n \log n)$ bits and messages. A de Bruijn Sequence $\beta_k$ is a sequence of $2^k$ bits with the property that each binary string of length $k$ occurs in $\beta_k$ exactly once as a cyclical substring [B46]. For each $k \geq 1$ there are many such sequences (see e.g. [E79]). From now on assume that $\beta_k$ denotes a fixed such sequence with the property that the the first bit of the sequence is marked.

For example, $\beta_k$ can be constructed as follows: start with $00^k 1$; bit $i$ ($k+1 \leq i \leq 2^k$) is one if the string of length $k$ formed by the bits $i-k+1, i-k+2, \ldots, i-1$ appended by a one does not appear yet in the sequence; otherwise bit $i$ is zero. The sequences for $k=1, 2, 3, 4$ are $01, 0011, 00111101, 000011101100101$ respectively.

We will use prefixes of $\beta_k$ to construct patterns recognizable with a small number of messages. Note that in the prefixes each new copy of $\beta_k$ starts with a marked bit. Let $\pi_{k,n}$, for $n \geq k$, be the first $n$ bits of $\beta_k$. For example, $\pi_{3,21}=0001110100100111010001$. The Algorithm $A(k,n)$ will recognize the pattern $\pi_{k,n}$ as well as cyclical shifts thereof. We will show that the algorithm requires $O(kn)$ messages and $O((k + \log n)n)$ bits. It only works if $2^k$ does not divide $n$. The minimum such $k$ might be small and in the worst case the choice of $k=\lceil \log n \rceil +1$ leads to a bound of $O(n \log n)$ messages and bits.

The algorithm will first check whether the input string fulfills a local condition for legality. For example, for Algorithm $X$ the pattern $00, 01$ and $10$ where legal and illegal was not. Let $\omega=\omega_1 \cdots \omega_n$ be a cyclical string of length $n$ representing the input of the $n$ processors. Then $\omega_i$ is legal w.r.t. $\pi_{k,n}$ if the $k$ bits to the left of $\omega_i$ appended with $\omega_i$ produces a string that occurs as a cyclical substring in $\pi_{k,n}$. If all bits of $\omega$ are legal then this string must be equal to $\pi_{k,n}$ or a close relative thereof. Note that the $k$ in the algorithms will be much smaller than $n$.

Lemma 8: If all $n$ bits of an input string $\omega$ are legal w.r.t. $\pi_{k,n}$, then some cyclical shift of $\omega$ is in the language $L=(\beta_k+\pi_{k,n}(mod2^k))^\star$. If in addition $n=O(mod2^k)$ then $\omega$ is in $\beta_k^\star$.

Proof: Let us first consider the case $n=O(mod2^k)$. Then $\pi_{k,n}$ is in $\beta_k^\star$ and any cyclical substring of $\pi_{k,n}$ of length $k$ uniquely determines the bit that is following the substring. Thus in this case the legality of $\omega$ implies that it is in $\beta_k^\star$.

If $n \neq O(mod2^k)$, then $\pi_{k,n}=(\beta_k)(n/2^k)\pi_{k,n}(mod2^k)$. Recall that $\pi_{k,n}(mod2^k)$ are the first $n(mod2^k)$ bits of $\beta_k$. All cyclical $k$-substrings of $\pi_{k,n}$ uniquely determine the next bit. There is only one exception to this: the last $k$ bits of $\pi_{k,n}(mod2^k)$. After these $k$ bits there are two choices for the next-bit: the first bit of $\beta_k$ (which is marked) or bit $(n(mod2^k))$+1 of $\beta_k$. From this it is easy to see that the legality of $\omega$ implies its membership in the language $L$. □

Algorithm $A(k,n)$:
For all $n$ processor in parallel do
A 1. Send your bit to the right neighbor and forward $k$ bits to the right neighbor.
A 2. After having received $k$ bits check whether your bit is legal w.r.t. $\pi_{k,n}$. If your bit is not legal, send a zero-message to the right and terminate with output zero.
A3. If you receive a zero-message forward it and terminate with output zero.

A4. If your input is marked initiate an X-counter message with count one. Otherwise forward the first X-counter message received after increasing its counter by one.

A5. If you initiated an X-counter then wait until you receive an X-counter. If its count is \( n \mod 2^k \), initiate Y-counter with count one. Otherwise forward the first Y-counter received after increasing its counter by one.

A6. If you initiated a Y-counter then wait until you receive a Y-counter. If its count is \( n \mod 2^k \), initiate a zero-message and terminate with result one. Otherwise, i.e. if you did not initiate a Y-counter, forward the message you receive and terminate with output zero if it was a zero-message and with output one if it was a one-message.

In Step A4 the processors with marked bits will start an X-counter. The lemma guarantees that at least one marked bit appears in \( \omega \) if the whole input is legal. Thus at least one X-counter message is initiated. Together with the assumption that \( n \neq 0 \mod (2^k) \) the lemma guarantees that there will be at least one X-counter ending with count \( n \mod 2^k \) and therefore at least one Y-counter will be initiated. If there is more than one Y-counter initiated then this will lead to a zero message and the output will be zero. Only in the case where all bits are legal and exactly one processor receives its own Y-counter with count \( n \) and the result of \( A(k, n) \) is one. In that case the input must be a cyclical shift of \( \pi_{k, n} \). If \( 2^k \) divides \( n \), either at least two or no Y-counters can be started in Step A5. Thus the algorithm either terminates with output zero or does not terminate, respectively, and in the latter case the input must be a cyclical shift of a pattern of the form \( \beta_k^+ \).

It is easy to see that all steps except Step A1 require \( O(n) \) messages. In Step A1, \( O(kn) \) messages are sent. Note that the \( O(n) \) counter messages requires \( O(n \log n) \) bits. The correctness proof is summarized in the following lemma which shows that the lower bound of Theorem 1 on the bit complexity of non-constant functions is tight. (Choose \( k = \lfloor \log n \rfloor + 1 \).)

Lemma 9: Let \( n \neq 0 \mod (2^k) \). The Algorithm \( A(k, n) \) recognizes \( \pi_{k, n} \) and cyclical shifts thereof in \( O(kn) \) messages and \( O((k+\log n)n) \) bits. \( \square \)

In Algorithm B we use for convenience a three letter alphabet, \( \{0, 1, \#\} \), and bits may be marked as before. Algorithm B first checks in Step B0 whether the input is of the form \( (#01)^{\log^* n} \). If so then the \( i \)-th processors after \# signs are of type \( i \), for \( 1 \leq i \leq \log^* n \). The processors with a \# sign are of type 0. Note that there are exactly \( n' = \frac{n}{1+\log^* n} \) processors of each type.

During the \( i \)-th phase of the loop the processors of type \( i \) will run a version of \( A(g_{i-1}, n) \) on their inputs. The steps in Algorithm A with the same numbers are related. Algorithm B is to work for arbitrary ring size \( n \). So it could be that \( 2^{g_{i-1}} \) divides \( n' \). This leads to
the definition \( l(n) \) which equals the minimum \( i \geq 1 \) such that \( g_i \) does not divide \( n' \). Note that \( l(n) \leq \log^* n \) since \( g_{\log^* n} \) does not divide \( n' \).

We will show that the algorithm recognizes the language \( L(n) \) which are all words of the form \((\# (0,1)^{\log^* n})^*\) such that: starting from some \# sign, for \( 1 \leq i \leq l(n) \), the bits of the type \( i \) processors form the pattern \( \pi_{g_i \cdot n} \). Note that from the definition of \( l(n) \) we know that \( \pi_{g_i \cdot n} \) is in \( \beta_{g_i} \) for \( 1 \leq i \leq l(n) \).

Algorithm \( B(n) \)
For all \( n \) processors in parallel do

\[ g_0 = 1 \]

For phase \( j = 1 \) to \( l(n) \) do

Begin

\[ g_i = 2^{g_{i-1}} \]

\[ B1. \] All processors with tokens send their bit to the right and wait until a message (denoted by \( M \)) arrives.

If you are not of type \( i \) then forward \( M \).

If you are of type \( i \) and have no token then

Append your input bit to \( M \) and forward it.

\[ B2. \] If you have a token then \( M \) consists of \( g_{i-1} \) bits. Determine if your bit is legal w.r.t. \( \pi_{g_i \cdot n} \). If your bit is not legal, send a zero-message to the right and terminate with output zero.

\[ B3. \] If you receive a zero-message forward it and terminate with output zero.

\[ B4. \] If your input is marked and you have a token initiate an \( X \)-counter message with count one.

Otherwise forward the first \( X \)-counter message received after increasing its counter by one if you are of type \( i \).

\[ B5. \] If you initiated an \( X \)-counter then wait until you receive an \( X \)-counter. It its count equals \( g_i \), then forward your token.

Otherwise delete your token.

End.

\[ B5'. \] If you initiated an \( X \)-counter in the last phase and received an \( X \)-counter with count \( n \pmod{g_{l(n)}} \), initiate \( Y \)-counter with count one.

Otherwise forward the first \( Y \)-counter received after increasing its counter by one.

\[ B6. \] If you initiated a \( Y \)-counter then wait until you receive a \( Y \)-counter. If its count is \( n \),
initiate a one-message and terminate with result one.
If you initiated a Y-counter and receive a counter \( <n \), initiate a zero-message, and terminate with result zero.
Otherwise, i.e. if you did not initiate a Y-counter, forward the message you receive and terminate with output zero if it was a zero-message and with output one if it was a one-message.

During phase \( i \), for \( 1 \leq i \leq (n) - 1 \), the algorithm checks whether the bits of the type \( i \) processors form the pattern \( \beta_{g_{i-1}} \). Note that each copy of \( \beta_{g_{i-1}} \) has length \( g_i \). In the phase \( l(n) \) together with steps \( B5' \) and \( B6 \), the processors of type \( l(n) \) check whether their input is a cyclic shift of \( \pi_{g_{l(n)-1}} \).

We will show that in each phase only a linear number of messages is required which leads to the overall bound of \( O(n \log^* n) \). In Step \( B2 \) we check whether the bits of the type \( i \) processors are legal. To do this the type \( i \) processors need to know the bits of the previous \( g_i \) processors of type \( i \). We have to refine Step \( A1 \) which required \( O(kn) \) messages (here \( k = g_{i-1} \)). The corresponding Step \( B1 \) will only require \( O(n) \) messages per phase.

In Step \( B1 \) only processors with tokens will send their bit to the right. By induction it is easy to see that at the beginning of phase \( i \) only type \( i \) processors have tokens and these tokens are distance \( g_{i-1}(1+\log^* n) \) apart. Note that this is true for phase 1 and in Step \( B5' \) of phase \( i \) the tokens that remain are distance \( g_i(1+\log^* n) \) apart. In Step \( B1 \) the processors with tokens start messages that travel until they reach the next processor with a token. On their way they gather all the bits of the type \( i \) processors (\( g_{i-1} \) of them). The paths of these messages do not overlap and thus Step \( B1 \) costs \( O(n) \) messages per phase. Thus we have:

**Theorem 3:** Algorithm \( B(n) \) recognized \( L(n) \) with \( O(n \log^* n) \) messages.
Acknowledgement: We would like to thank Gerhard Bu&trock for many useful discussions.

REFERENCES


