Gap Theorems for Distributed Computation

Shlomo Moran *
Manfred K. Warmuth **

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 $O(nlogn)$. 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.

* Department of Computer Science, the Technion, Haifa 32000, Israel. This research was done while the S. Moran was on leave at IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

** Department of Computer and Information Sciences, University of California, Santa Cruz, CA 95064. The research of M. K. Warmuth was supported by Faculty Research funds granted by the University of California, Santa Cruz.

1. 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 [ASW85, 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, $\Omega(nlgn)$ 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 $\Omega(nlgn)$ 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 [ASW85]). 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 [ASW85].

Let us contrast the anonymous ring with the model consisting of a chain of asynchronous processors where the head and tail processors of the chain are distinguished, or with a ring of asynchronous processors with a leader. If we assume unidirectional communication then any non-trivial function must require $O(n)$ 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 $c(n) \leq n^2$. The input is denoted as an n-bit word ω, the i-th bit ω_i of ω 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 ω contains a palindrome of $2\left\lceil\sqrt{c(n)}\right\rceil+1$ bits centered at the leader. Thus there is no gap for a chain of processors and for rings with a leader. Our gap theorem for anonymous rings clearly quantifies the price one has to pay for closing the chain to a ring and 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, if the size of the ring has a constant non-divisor. In the case where there is no such non-divisor, 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 $O(n\log^* n)$ messages.

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 $\Omega(nlgn)$ bits. There is a crucial difference between the model of [MS] 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 [ASW85] for the anonymous ring without a leader which is our model). Without the knowledge of the ring size the processors can only compute regular languages on a ring with a leader and nonregular languages require $\Omega(nlogn)$ 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 [ASW85], 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 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 $\Theta(nlogn)$.

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 [ASW85]. 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 for some alphabet I. 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).

In the next section 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 third and 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 $O(nlogn)$ bit complexity, respectively. The functions are defined by a recursive application of de Bruijn sequences.

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, take the function which equals one iff the distinct id's of the given ring are the n smallest id's.

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 $\Omega(nlogn)$.

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 4.1 of [ASW85]. 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* (infinite delay) or are *synchronized* (it takes exactly one time unit to traverse the link). For the below lemma 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 $0^z \cdot \tau$ for any τ, then AL requires at least $n \lfloor z/2 \rfloor$ messages for 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 now on the processor cannot change any more 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 $0^z \cdot \tau$. If $T \le \lfloor z/2 \rfloor$ 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. Let $\omega \in I^n$ be a word in S. Consider an execution of the algorithm AL with ω as input and all linked 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 message are received from the left). For $0 \le s \le t$ and for $1 \le i \le n$ we define $h_i(s)$, the *history* of processor i after s time units, to be the string $h_i(s) = m_i(1)L \cdots Lm_i(k)$, where L is a special symbol and $m_i(1),..,m_i(k)$ 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 $O(n)$ and therefore the below lemma implies that $\Omega(n \log n)$ bound.

Lemma 2: Let $H_1,..,H_k$ be k distinct words over an alphabet of size r. 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 might not be responsible for any H_i. The case $r=1$ is trivial and thus consider $r>1$. Assume the overall length of the H_i is minimized. Then in the corresponding tree all but one node have degree one and at least half of the nodes are leaves. The lemma is implied by the fact that the average path length for an r-ary with v leaves is at least $\log_r v$. Proofs of this fact are usually discussed in connection with the average case lower bound for the number of comparisons required for sorting (see e.g. [AHU83]). □

Outline of the proof of Theorem 1: An execution of AL is constructed for which one of the two lemmas implies the lower bound of $\Omega(n \log n)$ bits. In the first case a processor accepts and the input that contains $\log n$ consecutive zeroes. 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 $p_{1,1}, p_{2,1},..,p_{n,1}, p_{1,2},..,p_{n,k}$. 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 jth 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 ω^k be the input to C, where $p_{i,j}$ receives the letter ω_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 $0 \le s \le t$ the histories $h_{i,j}(s)$ and $H_{i,j}$ of the processor $p_{i,j}$ in C are defined similarly to the histories of the processors in R 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 $0 \le s \le t$. 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 that 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 $\bar{C}$ 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 $\bar{C}$ 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 $p_{n,k}$. $\bar{C}$ 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 $\bar{C}$ have the same history in the execution on C.

Proof: The first processor in $\bar{C}$, $p_{1,1}$, 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 processor p in C with $H_p=H$. □

Let the sequence of processors $\bar{C}$ defined above be $\bar{C}=(p_{i_1,j_1},..,p_{i_m,j_m})$, and let τ be the input word $\omega_{i_1} \cdots \omega_{i_m}$ of length m. Note that p_{i_l,j_l} gets $\omega_{i_l}=\tau_l$ as an input. We now run AL with all links synchronized except for the link $p_{i_m,j_m}-p_{i_1,j_1}$ $(=p_{n,k}-p_{1,1})$ which is blocked.

Lemma 5: In the execution of AL on $\bar{C}$ with input τ the history of processor p_{i_l,j_l} $(1 \le l \le m)$ of $\bar{C}$ is the same as the history of p_{i_l,j_l} of in the execution of AL on C with input ω^k. In particular, processor p_{i_m,j_m} $(=p_{n,k})$ of $\bar{C}$ accepts.

Proof: This follows from a simple induction of l using the way $\bar{C}$ 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 $1 \le l \le m$, the number of bits received by l distinct processors of $\bar{C}$ in the execution described above is at least $(l/4)\log_3(l/2)$.

Proof: Let L be a set of l processors in $\bar{C}$, and let $p \varepsilon L$. By Lemma 5, the history of p in the computation of $\bar{C}$ 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 τ' be the first n letters of $\tau \cdot 0^*$ and let $m'=min(\{m,n\})$. Consider a computation of AL on τ' in which the first m' processors of $\bar{C}$ have exactly the same history as the corresponding first m' processors in the computation of $\bar{C}$ on τ described above, and no message sent by the remaining processors is ever received. We distinguish two cases depending on the length m of $\bar{C}$.

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

Case $m > n - logn$: 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 $\Omega(nlogn)$.

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-trivial set S over I^n. Assume further that 0^n is not in S. Let ω be a word in S, and again consider a "synchronized" computation of AL on ω. The history of a processor p_i at time s in such a computation is a string $h_i(s)=d_i(1)m_i(1) \cdots d_i(k)m_i(k)$, 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 ω 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 ω^{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, $p_{2,1}$ and $p_{n-1,k}$ are blocked at time two, and in general at time s $(1 \le s \le t)$, the s leftmost and the s rightmost processors of D are blocked. Note that no message sent on the link $p'_{n,k}-p_{1,1}$ 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.

Lemma 3': Let $p_{i,j}$ $[p'_{i,j}]$ be the sth leftmost [rightmost] processor in D $(1 \le s \le t)$. Then $h_{i,j}(t)=h_i(s-1)$. That is, at the end of the above execution on D, the history of the sth 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 $p_{n,k}$. $\overline{C}$ $(\overline{C}')$ 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 $\overline{C}$ or in $\overline{C}'$ have the same history in the execution on 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 $\overline{C}$ or in $\overline{C}'$ by $(l/4)\log_4(l/2)$.

Let m be the length of D. In the case where $m \le n$ 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

$m \le n - \log n$ or not, one can complete the proof of Theorem 1' as before, provided that $m \le n$.

However if $m > n$ then one cannot proceed as in the unidirectional case. If we would cut the first m processors of D, then the last processor does not receive the proper messages from the right and the proof requires modification. For $b=1,\dots,k$, 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 ω^{2b}, and use it to define $\bar{D}_b$, $b=1,\dots,k$. Let m_b be the number of processors in $\bar{D}_b$. 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) $m_1 \le 2n$ (since D_1 has only $2n$ processors),
(ii) $m_k > n$ (by the assumption on $m=m_k$), and
(iii) $m_1 \le m_2 \le \cdots \le m_k$ (by inclusion).

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

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 a proof similar to that of Theorem 1, with m_b replacing m', implies the result. This completes the proof of Theorem 1'. □

Surprisingly the gap remains for rings with id's if the set of possible id's is large enough relative to n. To prove this, we define $eh_i(s)$, the *extended history* of the processor p_i at time s, to be a string $t_i(1)d_i(1)m_i(1)\dots t_i(k)d_i(k)m_i(k)$, where the $d_i(j)$s and the $m_i(j)$s are, as before, directions and messages, and the $t_i(j)$s are either I if the corresponding message was received by p_i, or O if it was sent by it.

Theorem 2: For a bidirectional ring of n processors with distinct id's, the bit complexity of 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 $o(n \log n)^\dagger$ computing a non-constant function and let Z be the set of possible id's, with $|Z| \ge n 2^{n^n}$. 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 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 $o(2^{n \log n}) = o(n^n)$. Denote $z \equiv_{AL} z'$ if $H_z = H_{z'}$. The relation $\equiv_{AL}$ partitions Z into at most $2^{|H|} = o(2^{n^n}) = o(\frac{|Z|}{n})$ 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. □

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 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

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

$O(nlogn)$ bit complexity, showing that the $\Omega(nlogn)$ bounds on the bit complexity given in the previous section are tight. All our algorithm are unidirectional. It is easy to derive symmetric bidirectional versions of them s.t. the message complexity increases by only a constant factor.

First we define a pattern which is based on the de Bruijn sequences [B46] that can be recognized in $O(nlogn)$ messages. The algorithm which achieves this bound is presented. It will be used $\log^* n-1$ times (each time with different parameters) in an algorithm that recognizes another 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. $g_i \geq n$. Note that $g_{i-1} \geq \log n$. A de Bruijn Sequence β_k is a sequence of 2^k bits with the property that each binary string of length k occurs in β_k exactly once as a cyclical substring [B46]. For each k there are many such sequences (see e.g. [E79]). During the discussion here, β_k will be a fixed de Bruijn sequence, that starts with 0^k.

We will use prefixes of β_k^* to construct patterns recognizable with a small number of messages. Let $\pi_{k,n}$, for $n \geq k$, be the first n bits of β_k^*. Assume we want to recognize the pattern $\pi_{k,n}$.

The algorithm will first check whether the input string fulfills a local condition for legality. Let $\omega=\omega_1 \cdots \omega_n$ be a cyclical string of length n. Then ω_i is *legal* w.r.t. $\pi_{k,n}$ if the $2k-1$ bits to the left of ω_i appended with ω_i produces a string that occurs as a cyclical substring in $\pi_{k,n}$. If all bits of ω 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 7: If all n bits of a string ω are legal w.r.t. $\pi_{k,n}$, then some cyclical shift of ω is in the language $L=(\beta_k+\pi_{k,n\,(mod\, 2^k)})^*$.

Proof: Let τ be the longest cyclical substring of ω occurring as a cyclical subword of some word in L. From the definition of legality it follows that either τ can be enlarged, i.e. it is not the longest such substring, or $\tau=\omega$. □

Note that the lemma implies that 0^k appears at least once in ω. At some point of the algorithm, the processors that are preceded by 0^k will take charge. The lemma guarantees that these processors always exist.

Our first algorithm is to recognize (output one) $\pi_{k,n}$ as well as cyclical shifts thereof. On all other inputs the output is zero. The algorithm requires $O(kn)$ messages and 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 $k=\lceil logn \rceil+1$ leads to a bound of $O(nlogn)$.

Algorithm $A(k,n)$:

For every processor in parallel do

A1. Send your bit to the right neighbor and forward $2k-1$ bits to the right neighbor.

A2. After having received $2k-1$ 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 you are succeeded by 0^k, 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 and receive an X-counter with count $n\,(mod\,2^k)$, 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 and receive a Y-counter with count 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.

The above lemma guarantees that there is either an illegal bit in ω or there is at least one processor succeeded by 0^k. 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 such 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 the result of $A(k,n)$ is one. In that case the input must be a cyclical shifts 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 β_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.

Lemma 8: 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 $O((k+logn)n)$ bits. □

We now apply the $A(k,n)$ altogether $n \log^* n - 1$ times in an Algorithm B that computes a non-constant function in $O(n \log^* n)$ messages. For convenience we use a three letter input alphabet: $\{0,1,\#\}$. With an encoding scheme two letters would suffice. We further assume that $\log^* n$ divides n. Otherwise $A(\log^* n, n)$ computes a non-constant function in $O(n \log^* n)$ messages, which implies the desired bound.

Algorithm B first checks whether the input is of the form $\#\{0,1\}^{\log^* n - 1})$. If so then it recognizes the following string or cyclical shifts thereof: starting from some #-sign the i-th bits following the #-signs form the string $\pi_{g_i,n'}$, where $1 \leq i \leq n \log^* n - 1$ and $n' = \frac{n}{\log^* n}$. An outline of algorithm B is given below:

Algorithm B (outline):

```
For every processor in parallel do
B1.  Send your input to your left neighbor and for-
     ward $\log^* n - 1$ inputs.
     If The number of # signs you have received is
     not exactly one, terminate with output zero and
     initiate a zero-message.
     (*At this point there are $\frac{n}{\log^* n}$ # signs in the
     input all of which are exactly $\log^* n$ apart.*)
B2.  $g_0=1$
     For phase $i=1$ to $\log^* n - 1$ do
         $g_i=2^{g_{i-1}}$
         Run $A(g_i,n')$ on the $i$-th processors fol-
         lowing the # signs.
         All other processors just forward messages
         without changing counts.
         If any processors receives a one- or a
         zero-message then this message is for-
         warded and the processor terminates.
```

The above outline of algorithm B has two draw backs: the first is that if, for some i, 2^{g_i} divides n', then the execution of phase i of B2 will never terminate. The second is that even if the algorithm terminates, step B2 has a high bit complexity: Note that $g_{\log^* n - 1} \geq logn$ and therefore $\Omega(n \log n)$ messages are sent during the last loop (in Step A1).

Both these problems are solved by introducing special *distributor* processors at each phase, which are the only processors that initiate Step A1 of that phase. In the first phase all the processors located to the right of the #-signs are distributors. At the end of phase i all processors to the right of the processors that initiated an X-counter and received an X-counter of count $2^{g_i} = g_{i+1}$ become distributors for phase $i+1$. Note that these distributors will be distance $g_{i+1} \log^* n$ apart. The distributors of phase $i+1$ start executing on the $i+1$ phase even though A5 and A6 of phase i might not be completed. In phase $i+1$ the $(i+1)$-st bits need to know the $2g_{i+1}-1$ previous $(i+1)$-st bits (Step A1). This expensive task can now be achieved by the distributors in a linear number of messages.

Since execution is asynchronous, different processors might not be in the same phase. In some phase a zero- or a one-message must be created. Before a one-message is created, a Y-counter messages travels around the ring and this counter message can check whether the input is of the required form. Note that the last phase will certainly cause a processor to terminate since $2^{g_{n\log^* n}-1} \geq n$ does not divide $n' = \frac{n}{n\log^* n}$.

In each phase only a linear number of messages are sent. Since there are $\log^* n$ phases the overall message bound is $n\log^* n$.

ACKNOWLEDGEMENT: We would like to thank Gerhard Buntrock for many useful discussions.

REFERENCES

[AHU83] A. V. Aho, J. E. Hopcroft, J. D. Ullman, "Data Structures and Algorithms," Addison-Wesley, 1983.

[ASW85] C. Attiya, M. Snir, and M. K. Warmuth, Computing on an Anonymous Ring, Technical Report UCSC-CRL-85-3, Department of Computer Science, University of California, Santa Cruz, November 1, 1985, a preliminary version appeared in the *Proc. of the Fourth Annual ACM Symposium on Principles of Distributed Computation, Minaki, Ontario, Canada, August 1985, pp. 196-203.*

[B46] N. G. de Bruijn, "A Combinatorial Problem," Koninklijke Nederlands Akademie van Wetenschappen, Proceedings, Vol. 49 (Part 2), 1946, pp. 758-764.

[B80] J. E. Burns, "A Formal Model for Message Passing Systems," Technical Report No. 91, Computer Science Department, Indiana University, Bloomington, In. 1980.

[DKR82] D. Dolev, M. Klawe and M. Rodeh. "An $O(nlogn)$ Unidirectional Algorithm for Extrema Finding in a Circle," *J. of Algorithms,* Vol. 3, No. 3,1982, pp. 245-260.

[E79] S. Even, Graph Algorithms, Computer Science Press, 1979.

[FLM85] M. J. Fischer, N. A. Lynch, and M. Meritt, "Easy Impossibility Proofs for Distributed Consensus Problems," *Proc. of the Fourth Annual ACM Symposium on Principles of Distributed Computation,* Minaki, Ontario, Canada, August 1985, pp. 59-70.

[H68] J. Hartmanis, "Computational Complexity of One-Tape Turing Machine Computations," *Journal of the ACM,* Vol. 15, No. 2, 1968, pp. 325-339.

[MZ85] Y. Mansour and S. Zaks, On the Complexity of Distributed Computations in a Unidirectional Ring With a Leader, TR 383, Computer Science Department, Technion, Haifa, Israel, November 1985, an extended abstract appears in the *Proc. of the Fifth Annual ACM Symposium on Principles of Distributed Computation, Calgary, Canada, August 1986.*

[P82] G. L. Peterson, "An $O(nlogn)$ Unidirectional Algorithm for the Circular Extrema Problem," Transactions on Programming Languages and Systems, Vol. 4, 1982, pp. 758-762.

[PKR84] J. Pachl, E. Korach and D. Rotem, "A new technique for proving lower bounds for distributed maximum-finding algorithms," *Journal of the ACM,* Vol. 31, No. 4, Oct 1984, pp. 905-918.

[T64] B. A. Trachtenbrot, "Turing computations with logarithmic delay" (In Russian), *Algebra i Logica,* Vol. 3, 1964, pp. 33-48, English Translation in University of California Computing Center, Technical Report, No. 5, Berkeley, California, 1966.