Information Processing Letters, 54(2):111--119, April 1995

Message Terminating Algorithms for Anonymous Rings of Unknown Size

Israel Cidon*and Yuval Shavitt
Electrical Engineering Dept.
Technion - Israel Institute of Technology
Haifa 32000, ISRAEL

Abstract

We consider a ring of an unknown number of anonymous processors. We focus on message terminating algorithms, i.e., algorithms that terminate when no more messages are present in the system but the processors may lack the ability to detect this situation. This paper addresses the design of deterministic and probabilistic algorithms that message terminate with the correct result. For this model we show:

- A deterministic ring orientation algorithm that employs symmetry breaking link markings and requires $O(n \log^2 n)$ bits for communication and $O(n)$ time. A probabilistic Las-Vegas version of this algorithm (that requires no link markings) has the same average communication and time complexities.
- A probabilistic Las-Vegas algorithm for partitioning an even size ring into pairs that requires $O(n \log n)$ communication bits and time.
- The impossibility of computing (via a Las-Vegas algorithm) most functions (the class of *nonsymmetric* functions) including: leader election, XOR and finding the ring size. The same technique can be applied to prove the impossibility of partitioning an odd size ring into a maximal number of pairs.

Keywords: Algorithms, Anonymous rings, message terminating algorithms, orientation.

*currently with Sun Microsystems Labs, Mt. View, CA 94043, USA

Information Processing Letters, 54(2):111--119, April 1995

1 Introduction

Ring networks have traditionally been used as simple frameworks for evaluating distributed computation and communication systems. The ring topology can be used to demonstrate the implication of symmetry on the performance and feasibility of certain distributed tasks.

In this work we consider rings of an unknown number of indistinguishable processors that communicate by messages sent over bidirectional links (*anonymous rings*). We use an asynchronous model in which message delivery time over a link is arbitrarily long but finite. Algorithms for such asynchronous systems are generally message driven in the sense that a processor acts (changes its state, sends messages, performs computations, etc.) only upon the reception of a message. At all other times the processor is idling. We focus on *message terminating* algorithms, i.e., the algorithm terminates when no more messages are present in the system (within buffers or in transit at the transmission lines) and no processor is at a state from which it may initiate a message. As was shown in [IR81, IR90], the processors may lack the ability to recognize the termination of the algorithm and it might be detected only by an outside observer. The stronger property of *processor termination* (PT) where at least one of the processors can detect the termination of the algorithm is not achievable for most non-trivial problems in anonymous rings of unknown size. Attiya, Snir, and Warmuth [ASW88] prove that no (deterministic) PT algorithm exists even for “simple” functions such as OR and AND. In fact, their proof can be easily extended for probabilistic PT algorithms as well.

Most of the algorithmic work in the area of anonymous networks has concentrated on models where some knowledge of the size of the network is known in advance (see [AS91, AAHK87, MW86]). In the absence of the ring size knowledge it was shown in [IR90] that functions such as computing the size of the ring or leader election can be computed by a probabilistic message terminating algorithm with some (arbitrarily small) error probability (Monte-Carlo algorithm). These results were extended in [SS89] for general topology networks.

In this paper we focus on message terminating algorithms for anonymous rings of unknown size that terminate correctly, either deterministically or with probability 1, while the average communication and time complexities are kept bounded. To the best of our knowledge this is the first time that non-trivial problems are solved under this model. However, most problems can not be solved under this model. We prove impossibility results for a large class of problems (the class of nonsymmetric problems). It is interesting to observe that if the model is changed to a known ring size, then some of the problems which are unsolvable in our model (such as XOR) can be solved as efficiently [ASW88] as the problems we solve here (such as orientation). In other words, there are cases where the partition between solvable and unsolvable problems (in our model) completely vanishes (even in terms of different solution complexities) under the known ring size model.

We present a deterministic orientation algorithm for a ring of unknown size that uses link markings (similar to [IJ93]) to break symmetry and has a communication complexity of $O(n \log^2 n)$ bits ($O(n \log n)$ messages) and time complexity of $O(n)$. Pachl [Pac87] shows a matching lower bound for the message complexity for rings of unknown size with distinct identities. For the link marking we assume the existence of a daemon that marks each link with '1' on one side and '0' on the other side. We show how this daemon can be replaced by coin tossing to yield a Las-Vegas orientation algorithm, i.e., an algorithm that terminates correctly with probability 1 and maintains bounded average communication and time complexities. Interestingly, the Las-Vegas orientation algorithm has the same average bit and time complexities as the deterministic one. Another Las-Vegas algorithm is presented here for partitioning an even size ring to neighboring pairs (maximum matching).

Itai and Rodeh [IR81, IR90] present a message terminating Monte-Carlo algorithm that computes the ring size in $O(rn^3)$ messages of $O(\log n)$ bits, and an error probability of $O(2^{-nr/2})$. Note, that if the error probability is taken to zero (or r is taken to infinity) the message and time complexities approach infinity for any n. Other distributed Monte-Carlo algorithms can be found in [AS91, MA89].

We present an impossibility result for a class of tasks, termed *nonsymmetric* problems, by showing that they cannot be solved under the Las-Vegas model. Two of the problems in this class were shown to be unsolvable in the past: finding the ring size by Itai and Rodeh [IR81] and solitude detection (leader election) by Abrahamson et al. [AAHK90]. Other examples of nonsymmetric problems (which are the majority of the problems) are XOR and partitioning an odd size ring into a maximum number of pairs (maximum matching).

2 Orientation Algorithms

Consider a ring of unknown number of anonymous processors. Each processor (node) has a local notion of a left and a right link which is termed the *orientation* of the node. Two neighboring nodes have the same orientation if the link connecting them is considered to be right at one of them and left at the other. Throughout the paper when we use directions, left or right, they are given according to the local orientation of the node we refer to or according to a common local orientation of a referred group of nodes. Our aim is to coordinate all the processors to have the same orientation. The fact that processors are anonymous creates a symmetry that prohibits a deterministic solution (see [Ang80]). We overcome this impossibility by employing symmetry breaking link markings that enable neighboring processors to act deterministicly. In section 2.4 we show how the link markings can be avoided and replaced by coin tossing.

In the course of the algorithm each node i is a member of a *segment* of consecutive nodes with the same orientation and keeps a local variable called $length_i$ that specifies its relative position (left to right) in the segment. The leftmost node in the sequence (the tail) has $length = 1$, its right neighbor has $length = 2$, etc. Therefore, a segment is a sequence of adjacent nodes with the same orientation that satisfies: 1) The tail has $length = 1$. 2) If $length_i \neq 1$ then $length_i = length_{left(i)}+1$. 3) For the right most node (the head): $length_{right(i)} \neq length_i+1$ OR $orientation_{right(i)} \neq orientation_i$. The *size* of a segment is defined as the *length* of its head.

2.1 General Description.

When a node wakes up, it determines an arbitrary orientation, forms a single node segment by setting *length=1*, and sends the message RIGHT(1) over its right link. Upon the reception of a RIGHT(k) message from the left neighbor, the node does nothing except for recording the value $k+1$ in the variable *l_from_left*. Consequently, if all nodes initially decide on the same orientation, each node will receive a RIGHT(1) message over its left link and the algorithm terminates. However, if a node (which is in general the head of a segment) receives a RIGHT message over its right link, it learns that the node on its right has a conflicting orientation, and one of them must flip its orientation. A contest to resolve which node changes orientation (the loser) will be described shortly. The node that lost the contest reverses its orientation and joins the new segment by setting its variable *length* to be one higher than that of the winning node. After joining the new segment, the loser becomes the new head of the segment and spearheads the algorithm by sending a RIGHT message with the new length to its new right neighbor.

The winner of the contest is the node heading the longer segment. In case of a tie, a symmetry breaking marking for each link dictates the winner. One exception for this procedure is when the node has *length=1*. As was previously described, a node with *length=1* that receives a RIGHT(k) message on its left link, saves the length ($k+1$) of the segment on its left in the variable *l_from_left*. In this case (only), if a segment from its right will try to capture it, the node might join the segment on its left (and will become its head). The decision which segment to join depends on the length of the two conflicting segments.[1] The algorithm eventually stops when all RIGHT messages arrive over left links.

2.2 Formal Description.

Each node uses the following variables: **length** – a node's position in a segment (from the left). **l_from_left** – used only when *length=1*, and records the potential position of the node in the segment on its left, in case it will join it. **state** – the state of the node, *asleep* or *awake*.
Messages that are sent by the algorithm: **WAKE_UP** – an initialization message received from a higher level. **RIGHT(l)** – a message sent by a node over its right link, that contains its position in its segment (from left).

[1] This can be viewed as if the node queues the message from the left until it is captured by the segment from the right and then changes orientation and reacts to the message from its new right that causes it to change orientation again. In this scenario one extra message is sent. We term this version "the basic algorithm" and use it in some of the following proofs.

Information Processing Letters, 54(2):111--119, April 1995

The algorithm is given in Figure 1. Initially, all nodes have $state = asleep$.

2.3 Correctness and Complexity

First, we show that the algorithm always terminates with a complete orientation of the ring. Then we analyze the message and time complexities of the algorithm. In order to save space we only state the main theorems and lemmas and omit some of the proofs. The missing parts can be found in [CS91].

Theorem 1 *[CS91]*
The algorithm terminates within finite number of messages. Upon termination, all nodes are awake and have the same orientation.

Lemma 1 *[GK82]*
Let $F(m)$ be a discrete function with the following recursive definition:

$$\begin{aligned} F(1) &= 1 \\ F(m) &= \max_{1\le i<m}\{F(i)+F(m-i)+\min\{i,m-i\}\} = \max_{1\le i\le m/2}\{F(i)+F(m-i)+i\} \end{aligned}$$

then for $m \geq 1$

$$F(m) = \begin{cases} 2F(\frac{m}{2})+\frac{m}{2} & \textit{for even } m \\ F(\frac{m-1}{2})+F(\frac{m+1}{2})+\frac{m-1}{2} & \textit{for odd } m \end{cases}$$

Lemma 2 *[CS91]*
A. For all natural k, $F(m) = m(1+\frac{1}{2}\log_2 m)$ where $m = 2^k$.
B. $F(m) \leq m(1+\frac{1}{2}\log_2 m)$ for all $m > 2$.

Lemma 3 *$F(m)$, as defined in lemma 1, is the maximum number of messages[2] that may be sent for creating a segment of size m.*

Proof:

Let $G(m)$ be the maximum number of messages sent while a segment of size m is formed. Clearly $G(1) = 1$, since segments of size 1 are only formed when a node wakes up and sends a single message.

A segment of size m, $m > 1$ can be formed in one of the following three ways.

A segment of size $m-1$ sends a RIGHT message that wakes up a node which is *asleep*. That node sends one message as it joins the segment. The number of messages sent in this scenario is at most $G(m-1)+1$.

[2] All the messages that are sent by the nodes of the segment from the moment they wake up until (and including) the RIGHT(m) message is sent are counted.

Information Processing Letters, 54(2):111--119, April 1995

A segment of size i and a segment of size $m-i$ exchange RIGHT messages and the bigger one captures the other. The number of messages sent in this scenario is at most $G(i)+G(m-i)+\min\{i,m-i\}$. This is true since each processor in the captured segment changes orientation and sends a single message (lines $\langle R.8\rangle$ - $\langle R.10\rangle$ in Figure 1). Note that if a segment of size $m+1-i$ captures a segment of size i except for its leftmost node (which joins its left side segment instead), the number of messages for forming a segment of size m takes only $G(m+1-i)+G(i)+i-1$. However, this situation should be viewed as if the segment grows to length $m+1$ and shrinks back, and indeed, the segment will eventually be captured by the bigger one from the left.

A segment of size $m-1$ is growing when the tail of the segment to its right (that has the same orientation) decides to join him (instead of joining the smaller winning segment on its right), i.e., a segment of size one with the same orientation merges with it. The number of messages sent in this scenario is at most $G(m-1)+G(1)+1$

Therefore, we can bound $G(m)$ by the maximum of the above results:

$$\begin{aligned} G(1) &= 1 \\ G(m) &\leq \max\{[G(m-1)+1], \max_{1\leq i<m}\{G(i)+G(m-i)+\min\{i,m-i\}, [G(1)+G(m-1)+1]\} \end{aligned}$$

which yields

$$G(m) \leq \max_{1\leq i<m}\{G(i)+G(m-i)+\min\{i,m-i\}\}.$$

Therefore, $G(m)\leq F(m)$. □

Theorem 2 *The algorithm for a ring of size n terminates after no more than $n(1+\frac{1}{2}\log_2 n)$ messages are sent.*

Proof:

By lemma 3, $F(m)$ is an upper bound for the number of messages exchanged to form a segment of size m. Assume that the algorithm terminates when there are $k\leq n$ segments of sizes $m_1,\ m_2,\ \ldots,\ m_k$, where $m_1+m_2+\cdots+m_k=n$. If $k=1$ the result follows. If $k>1$ then the total number of messages sent is less or equal than $F(m_1)+F(m_2)+\cdots+F(m_k)\leq m_1(1+\frac{1}{2}\log m_1)+\cdots+m_k(1+\frac{1}{2}\log m_k)\leq n+\frac{1}{2}(m_1+m_2+\cdots+m_k)\log(m_1+m_2+\cdots+m_k)=n(1+\frac{1}{2}\log_2 n)$. □

For the purpose of time complexity analysis, we assume that a message is delivered after at most one time unit. In the basic algorithm, each node sends a message when it wakes up. We call the situation where two nodes send messages to each other on the same link a *conflict*. It is important to notice that a conflict can be created only when a node wakes up. After a conflict is created, one of the nodes changes its orientation and sends a message on its other link. This can lead to a conflict on that link and we shall say that the conflict *moves* one hop. Otherwise, we say that the conflict was resolved.

Information Processing Letters, 54(2):111--119, April 1995

Lemma 4 *[CS91]*
In the basic orientation algorithm, delaying an incoming message, cannot cause any message to be sent earlier from this node.

Lemma 5 *If all nodes are awakened together, after k time units there are no conflicts between segments that are both of size less than k.*

Proof:

Using lemma 4, for the the worst case, we assume that messages are always delayed by one time unit.

We prove the lemma by an induction on the integer k. After one time unit, all segments are of length 1, and the lemma holds. We assume that the hypothesis holds for all values less or equal to $k-1$, and prove for k.

Assume that at time k there are two conflicting segments of sizes $s_1 \geq s_2$, respectively. If the two segments were already at conflict at time $k-1$, then at that time one of them had a size of $s_1 - 1$ and the other had a size of $s_2 + 1$ (at time k $s_1 > s_2$ for this case). Given the hypothesis at time $k-1$, it must be that $s_1 \geq k$. If the two segments were not in a conflict at time $k-1$ then at least one of them has been in a conflict at time $k-1$ with a segment of size 1 (in order for them to be in a direct conflict at time k). Therefore, by the hypothesis either $s_1 - 1 \geq k-1$ or $s_2 - 1 \geq k-1$. □

Corollary 1 *If all nodes are awakened together, the algorithm terminates after no more than n time units.*

A simple addition to the algorithm enables termination for a ring of size n after no more than $\lceil \frac{3n}{2} \rceil$ time units. Every node that receives a message for the first time (wakes up) sends immediately a wake-up message on the other link (on both links). It is clear that after $\lceil \frac{n}{2} \rceil$ time units all the processors are awakened, and the extra cost is only $2n$ messages. Using lemma 4, from this time on the algorithm cannot terminate later than an algorithm that is executed over an identical ring where all the nodes are awakened together (by lemma 5, it takes at most n time units). Thus, the algorithm terminates after $\lceil \frac{3n}{2} \rceil$ time units.

Another simple version of the algorithm suggested by one of our anonymous reviewers, performs the above orientation algorithm without the use of the *length* field. The link markings are used to determine the orientation of the links. Two types of ties are possible: two links that point to a node (*sink*) or two links that point away from a node (*source*). A sink node chooses an arbitrary direction, and sends a message to flip the links in that side of it until the next source node. This algorithm improves the bit complexity by a factor of $logn$ but possesses a time complexity of $O(n \log n)$.

Information Processing Letters, 54(2):111--119, April 1995

2.4 A Las-Vegas Orientation Algorithm

The orientation algorithm presented above, uses a deterministic function to break symmetry among adjacent nodes. Consequently, it requires some predefined consistent input for each pair of neighboring nodes. These predefined markings can be replaced by a probabilistic tie breaking procedure. In order to break symmetry among neighbors, each node tosses a fair coin and sends the result to its competing neighbor. If they both toss the same value, another round of coin tossing is performed, until exactly one of the nodes gets '1'. The expected number of rounds for such a contest process is $\sum_{i=1}^{\infty} i \cdot (\frac{1}{2})^i = 2$. Therefore, the expected message complexity for the probabilistic orientation algorithm remains $O(n \cdot \log n)$. Similar to its deterministic counterpart, the probabilistic algorithm is partially correct, i.e., it outputs the correct result for every finite execution [IR81]. It message terminates with probability 1, and therefore it constitutes a message terminating Las-Vegas algorithm.

3 Partitioning an Even Size Ring into Pairs

We assume a bidirectional un-oriented ring with an unknown but even number of anonymous processors. We present a maximum matching algorithm for this ring.

The algorithm operates in phases: In each phase, a node sends and receives a single message over both links. (We delay messages which arrive out of phase until the next phase starts, and there can be at most one such outstanding message per node.) Nodes are either *single* or *married*. Only *single* nodes initiate the communication of the next phase. Two messages are used: *invite* and *reject*. At the beginning of a phase (or upon wake-up) a *single* node randomly selects one of its links, marks it as *candidate*, sends an *invite* message over that link and sends a *reject* message over the other link. If it receives an *invite* message over the *candidate* link, it marks the link as a *spouse* and becomes *married*. Otherwise, it waits until it has received a message over both links and starts a new phase. A *married* node only forwards the messages it gets to the opposite direction. In the case that both messages of the same phase are *invite*, the *married* node swaps its *spouse*, i.e., it removes the *spouse* marking from one of the links and marks the other as *spouse*. The change of the *spouse* marking shifts the matches of all the couples between two single nodes and repartitions these nodes and the two single nodes into neighboring couples. If all nodes are *married* after a phase completes, the algorithm terminates as no processor initiates a new phase.

We turn to the cost analysis of the presented algorithm. At each phase a single node becomes married with probability 1/2. Therefore, on the average half of the single nodes become married after each phase. Consequently, it is easy to show that the average number of phases is logarithmic in the number of nodes. Each phase requires exactly one message over every link in each direction, which totals to $O(n \cdot \log n)$ messages. Since there are only two message types (*invite* and *reject*), the communication bit complexity is also $O(n \cdot \log n)$.

Information Processing Letters, 54(2):111--119, April 1995

The above algorithm for partitioning an even size ring to pairs can be generalized to partitioning a ring of size km to m groups of k neighboring nodes.

3.1 Formal Description.

As in the previous section each node has a local orientation. We denote the left (right) link by the Boolean value '0' ('1'). Consequently, if the Boolean variable x contains the left (right) link identity then its complement $\bar{x}$ holds the opposite link identity.

The algorithm is given in figure 2.

The following procedures are used by the algorithm: *Receive*(m,l) – Writes in m the type of message that is at the head of the queue l and removes the message from the queue. If the queue is empty *Receive* first waits for a message arrival. *Random* – returns a random Boolean value (with equal probabilities).

Each node uses the following variables: *cs* – the current *candidate/spouse* link. *state* – the state of the node, *single* or *married*. $m1$ and $m2$ – the type of the received messages at queues 1 and 2. Initially all nodes have $state = asleep$.

Messages that are sent or received by the algorithm: **WAKE_UP** – an initialization message received from a higher level. *invite*, *reject* – as explained above.

4 Impossibility Results

Let $f : \Sigma^* \to T$ be a function computed on anonymous rings defined over some alphabet Σ. f is *symmetric* if for every string $s \in \Sigma^*$ and for every natural k, $f(s) = f(s^k)$. Otherwise f is *nonsymmetric*. (It is easy to realize that most functions f are nonsymmetric.)

An algorithm is *partially correct* if it outputs the correct result for every finite execution. Itai and Rodeh [IR81] prove the impossibility of calculating the ring size when the processors are anonymous, and the algorithm is partially correct. In the sequel we shall extend this proof in two directions. First we prove that the ring size cannot be calculated by algorithms that have bounded bit complexity and are partially correct with probability 1, where in [IR81] algorithms that err with probability 0 were not considered. Second we generalize this to all *nonsymmetric* functions.

Examples for *nonsymmetric* functions are leader election, XOR and computing the ring size. For XOR, $f(s) \neq f(s^k)$ whenever s contains an odd number of 1s and k is even. For computing the ring size, $f(s)$ never equals $f(s^k)$ if $k > 1$. For leader election there are no input values (except for the size) and the function return value is a bit that tells each processor whether it is the leader, it is clear that the vector $\vec{f}(s^k)$ never equals the concatenation of the k vectors of $\vec{f}(s)$.

The proof is constructed for a synchronous algorithm and thus holds for asynchronous algorithms as well.

Theorem 3 *There is no algorithm for computing a nonsymmetric function in a ring of unknown number of anonymous processors that is partially correct with probability 1 and its average bit complexity is bounded.*

Proof:

We assume to the contrary that such an algorithm exists and show a contradiction. Let $\mathcal{A}$ be such an algorithm for computing the function f in a ring with an unknown number of anonymous processors, and let E be the average number of bits sent by the algorithm. We examine a ring with n processors $(p_1,\ p_2,\ \ldots, p_n)$ with inputs $s = (s_1,\ s_2,\ \ldots, s_n)$. Let $f_i(s)$ be the value that processor p_i holds when the algorithm (message) terminates for inputs s, let $\vec{f}(s) = (f_1(s),\ f_2(s),\ \ldots,\ f_n(s))$, and let $\vec{f}^k(s) = (\vec{f}(s),\ \vec{f}(s),\ \ldots,\ \vec{f}(s))$ be the concatenation of k output vectors $\vec{f}(s)$. Select s and k s.t. $\vec{f}^k(s) \neq \vec{f}(s^k)$, where $\vec{f}(s^k)$ is the output vector for a ring of size kn with inputs s^k (k concatenations of the vector s). We select an execution[3] $\mathcal{R}$ of positive (greater than zero) probability that terminates after a finite number of bits were sent, and yields $\vec{f}(s)$ as an output vector. We now show that such $\mathcal{R}$ with probability greater than zero exists. By definition $E = \sum_{i=1}^{\infty} ip(i)$, where $p(i)$ is the probability that $\mathcal{A}$ terminates after i bits were sent. Applying the Markov inequality (see [Ros85]) we get $\sum_{i \geq 2E} p(i) \leq \frac{1}{2}$, and thus $\sum_{i \leq 2E} p(i) \geq \frac{1}{2}$. Since the number of executions that uses at most $2E$ bits is finite (recall that we are dealing with a synchronous model here) we can deduce that at least one such execution has a positive probability termed ε.

Examine a system of k rings each of n nodes and an input vector s. All the rings are stochastically independent. Since the probability space of this system is the Cartesian product of k identical probability spaces of the ring discussed before, the probability for $\mathcal{R}$ to be executed simultaneously at all the rings is ε^k.

Now cut all the above rings at the same place, say between processors p_i and p_{i+1}. Then, connect all the k strips into a ring of kn processors such that every processor p_i of ring j is connected to processor p_{i+1} of ring $j+1$ (modulo k): $(p_{i+1}^1,\ \ldots,\ p_n^1,\ p_1^1,\ \ldots,\ p_i^1, p_{i+1}^2,\ \ldots,\ p_i^2, p_{i+1}^3,\ \cdots, p_i^k)$. Since the processors are anonymous there is no way for a processor to tell if it is in the system of k rings that each one of them has $\mathcal{R}$ executed at, or whether it is in a ring of kn nodes that has k 'copies' of $\mathcal{R}$ executed at its k sections. The probability space of the concatenated ring and the k ring system is identical. All the events in the probability space that cause simultaneous execution in the k ring system cause also simultaneous execution of k copies of $\mathcal{R}$ at the concatenated ring and yields $\vec{f}^k(s) = (\vec{f}(s),\ \vec{f}(s),\ \ldots,\ \vec{f}(s))$ as output. These events are selected with the same probabilities at both systems. Therefore, the probability to get $\vec{f}^k(s)$ as output in the concatenated ring is at least ε^k. □

[3] An execution is defined by the messages that are sent (received) and by their transmission (reception) times.

A direct corollary of this theorem is that deterministic algorithms for nonsymmetric functions do not exist.

The rationale for our restriction to algorithms with bounded average bit complexity can be demonstrated by the following simple algorithm. Each node selects a real number in the range (0..1). Since the probability for two nodes to choose the same real number is zero the ring's nodes possess a unique identities with probability 1, a model that was studied in depth in the literature (see [PKR84], [Ray88, Ch. 2, Sec. 5], and the references therein). Leader election can now be easily performed with $O(n \log n)$ messages but each one of them carry an infinite number of bits.

5 Concluding Remarks

We presented message terminating algorithms for the problems of ring orientation and partitioning an even size ring into pairs. An algorithm for the problem of General Pattern Searching (GPS) can be found in [CS91]. Simple implementation of this algorithm can be used to compute the functions OR and AND (a search for a '1' or a '0', respectively) with $O(n)$ bit and time complexity.

Some problems are more difficult to describe in terms of functions. However our impossibility result still holds for such problems if they possess an asymmetric behavior. Such examples are: partitioning a ring of size $n \neq km$ to a maximal number of groups of k neighboring nodes; evaluating the size of the longest sequence of consecutive '1', where in the case that all inputs are '1' this translates into finding the ring size (which was also proved in [IR81]).

Acknowledgment

The authors are thankful to the anonymous reviewers for their constructive comments and suggestions that helped to improve the presentation of the paper.

References

[AAHK87] Karl Abrahamson, Andrew Adler, Lisa Higham, and David Kirkpatrick. Randomized function evaluation on a ring. In J. van Leeuwen, editor, *Proceedings, 2nd International Workshop on Distributed Algorithms*, pages 324–331, July 1987. Lecture Notes in Computer Science, Vol. 312.

[AAHK90] Karl Abrahamson, Andrew Adler, Lisa Higham, and David Kirkpatrick. Optimal algorithms for probabilistic solitude detection on anonymous rings. Technical report TR 90-3, University of British Columbia, 1990.

[Ang80] Dana Angluin. Local and global properties in networks of processes. In *12th annual ACM symp. on theory of computing*, pages 82–93, April 1980.

[AS91] Hagit Attiya and Marc Snir. Better computing on the anonymous ring. *Journal of Algorithms*, 12(2):204–238, June 1991.

[ASW88] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on the anonymous ring. *Journal of the ACM*, 35(4):845–875, 1988.

[CS91] Israel Cidon and Yuval Shavitt. Message terminate algorithms for rings of unknown size. EE Pub. 793, Technion - Israel Institute of Technology, Dept. of Electrical Engineering, Haifa 32000, ISRAEL, August 1991.

[GK82] Daniel H. Greene and Donald E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhauser, second edition, 1982.

[IJ93] Amos Israeli and Marc Jalfon. Uniform self-stabilizing ring orientation. *Information and Computation*, 104(2):175–196, June 1993.

[IR81] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. In *Proceedings of the 22nd annual IEEE symp. of fundations of computer science (FOCS)*, pages 150–158, 1981.

[IR90] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, September 1990.

[MA89] Yossi Matias and Yehuda Afek. Simple and efficient election algorithms for anonymous networks. In *Proceedings of the third international workshop on Distributed Computing (WDAG)*, pages 183–194. Springer-Verlag, September 1989. LNCS 392.

[MW86] Shlomo Moran and Manfred K. Warmuth. Gap theorms for distributed computation. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 141–150, 1986.

[Pac87] Jan K. Pachl. A lower bound for probabilistic distributed algorithms. *Journal of Algorithms*, 8:53–65, 1987.

[PKR84] J. Pachl, E. Korach, and D. Rotem. Lower bounds for distributed maximum-finding algorithms. *Journal of the ACM*, 31(4):905–918, 1984.

[Ray88] Michel Raynal. *Distributed Algorithms and Protocols*. Wiley, 1988. Translated from French.

[Ros85] Sheldon M. Ross. *Probabilities Models*. Academic Press, third edition, 1985.

[SS89] Baruch Schieber and Marc Snir. Calling names on nameless networks. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 319–328, 1989.

Information Processing Letters, 54(2):111--119, April 1995

for WAKE_UP
$\langle W.0 \rangle$ set $state \leftarrow awake$; $l\_from\_left \leftarrow -1$
$\langle W.1 \rangle$ set $length \leftarrow 1$
$\langle W.2 \rangle$ Send RIGHT($length$) on right link
for RIGHT(l) /* A message is received with parameter l */
$\langle R.0 \rangle$ if $state = awake$ then
$\langle R.1 \rangle$ if received on right link then
$\langle R.2 \rangle$ if ($l > length$) OR ($l = length$ AND NOT SYM_BREAK(right link)) then
$\langle R.3 \rangle$ if (($length = 1$) AND ($l\_from\_left > l$ OR
                        ($l\_from\_left = l$ AND SYM_BREAK(right link)))) then
            /* join the long segment at your left */
$\langle R.4 \rangle$ $length \leftarrow l\_from\_left$
$\langle R.5 \rangle$ Send RIGHT($length$) on right link
$\langle R.6 \rangle$ end
$\langle R.7 \rangle$ else /* the contest is lost - join the segment at your right */
$\langle R.8 \rangle$ change orientation
$\langle R.9 \rangle$ $length \leftarrow l + 1$
$\langle R.10 \rangle$ Send RIGHT($length$) on right link
$\langle R.11 \rangle$ end
$\langle R.12 \rangle$ else /* A message is received on left link */
$\langle R.13 \rangle$ $l\_from\_left \leftarrow l + 1$
$\langle R.14 \rangle$ else /* node is asleep */
$\langle R.15 \rangle$ set link on which the message is received to be the left one
$\langle R.16 \rangle$ $length \leftarrow l + 1$
$\langle R.17 \rangle$ Send RIGHT($length$)
$\langle R.18 \rangle$ end

Figure 1: The Orientation Algorithm – A Formal Description

Information Processing Letters, 54(2):111--119, April 1995

for WAKE_UP or message reception and $state = asleep$

$\langle W.1 \rangle$ $cs \leftarrow Random$

$\langle W.2 \rangle$ send $invite$ on link cs

$\langle W.3 \rangle$ send $reject$ on link $\bar{cs}$

$\langle W.4 \rangle$ $state \leftarrow single$

for a message arrival on link l

$\langle M.1 \rangle$ if $state = asleep$ then

$\langle M.2 \rangle$ do WAKE_UP procedure

$\langle M.3 \rangle$ if $state = married$ then

$\langle M.4 \rangle$ Receive($m1$, l)

$\langle M.5 \rangle$ Send($m1$, l)

$\langle M.6 \rangle$ Receive($m2$, $\bar{l}$)

$\langle M.7 \rangle$ Send($m2$, $\bar{l}$)

$\langle M.8 \rangle$ if $m1 = m2 = invite$ then

$\langle M.9 \rangle$ $cs \leftarrow \bar{cs}$

$\langle M.10 \rangle$ if $state = single$ then

$\langle M.11 \rangle$ receive($m1$, cs)

$\langle M.12 \rangle$ receive($m2$, $\bar{cs}$)

$\langle M.13 \rangle$ if $m1 = invite$ then

$\langle M.14 \rangle$ $state \leftarrow married$

$\langle M.15 \rangle$ else

$\langle M.16 \rangle$ $cs \leftarrow Random$

$\langle M.17 \rangle$ Send $invite$ on link cs

$\langle M.18 \rangle$ Send $reject$ on link $\bar{cs}$

Figure 2: The Partitioning Algorithm – A Formal Description