Leader Election and Sorting in Anonymous Asynchronous Rings [1]

Paola Flocchini

School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada

E-mail: flocchin@site.uottawa.ca

Evangelos Kranakis

School of Computer Science, Carleton University, Ottawa, Canada

E-mail: kranakis@scs.carleton.ca

Danny Krizanc

Mathematics Department, Wesleyan University, Middletown, CT, U.S.A.

E-mail: dkrizanc@wesleyan.edu

Flaminia L. Luccio [2]

Dipartimento di Scienze Matematiche, Università degli Studi di Trieste, Trieste, Italy

E-mail: luccio@dsm.univ.trieste.it

Nicola Santoro

School of Computer Science, Carleton University,Ottawa, Canada

E-mail: santoro@scs.carleton.ca

In an anonymous ring of n processors, all processors are totally indistinguishable except for their input values. These values are not necessarily distinct, i.e., they form a *multiset*, and this makes many problems particularly difficult. We consider the problem of distributively *sorting* such a multiset on the ring, and we give a complete characterization of the relationship with the problems of leader election for vertices and edges. For Boolean input values and prime n, we also establish a lower bound, and a reasonably close upper bound on the message complexity valid for sorting and leader election.

Key Words: Distributed Computing; Sorting; Leader Election; Multisets; Anonymous Ring.

[1] A preliminary version of this paper has been presented at the *IEEE International Parallel and Distributed Processing Symposium* [7].

[2] Correspondence may be sent to Flaminia L. Luccio, Dipartimento di Scienze Matematiche, Università degli Studi di Trieste, via Valerio 12/1, 34127 Trieste, Italy. Phone: +39 040/5582610 Fax: +39 040/5582636.

1. INTRODUCTION

A large body of research has been dedicated to the study and analysis of computing in an *asynchronous anonymous ring*: a ring network where all processors (or vertices) are totally indistinguishable except for their input values, and communication delays (which include transmission, queuing, and processing delays) are finite but unpredictable. This is due to the fact that asynchrony and anonymity render the network computationally weak, and, at the same time, the symmetry of the ring structure renders the resolution of most problems computationally non-trivial. Hence it is an ideal setting to study the complexity of problems and the relationship among them.

Let $R = r_0, \ldots r_{n-1}$ be an anonymous asynchronous ring with n processors (or vertices). Initially, each vertex r_i of the ring is assigned a value s_i from a totally ordered set $\mathcal{V}$. The input values are not necessarily distinct, and thus form a *multiset* $S = \{s_0, s_1, \ldots, s_{n-1}\}$. Let $\delta(S)$ denote the cardinality of the corresponding set, (i.e., the number of distinct elements of $\mathcal{V}$ in S); clearly, $\delta(S) \leq n$.

Computations in non-anonymous rings have been extensively studied in the literature and problems such as: leader election, edge election, minimum and maximum finding, sorting, topology recognition have been solved and analyzed. The motivation of these problems are well known. See [9] for an extensive treatment of the subject.

The case $\delta(S) = n$ corresponds to the case when S is actually a set, i.e., each vertex has a distinct input value. In this case, the network is not anonymous since the distinct values allow to distinguish among the vertices.

For the case $\delta(S) < n$, i.e., when S is not a set, very little is known. Most of the existing results focus on Boolean multisets (i.e., $\delta(S) \leq 2$) and study the problem of computing Boolean functions [2, 3, 4, 5, 6, 8, 10]. The remarkable papers [12, 14] address the leader election and other related problems for general topologies. Some of the results of [12, 14] are related to ours, but due to the generality of that model, the communication complexities obtained in [12, 14] are very high (in fact, studying complexity is not the main issue of those papers). Instead we will provide some efficient algorithms.

In this paper we consider the problem of *sorting* the input values, and other related problems in anonymous asynchronous rings with $\delta(S) < n$.

Solving the *multiset sorting problem* (*MSP*) means that, at the end of the computation, the input values are placed on the ring so that starting from some vertex and proceeding in only one direction, the values are encountered in order (see Figure 1); the choice of the direction and of the vertex is not pre-determined. We study the multiset sorting problem and investigate its relationship with three election problems.

The *vertex election problem* (*VEP*) consists in starting from a situation where the network is anonymous and ending in a situation where a vertex is distinguished from the others, i.e., is a leader. Analogously, in the *edge election problem* (*EEP*), an edge must become distinguished from all others, and identified as the leader. The *general election problem* (*GEP*) is the problem of electing, if possible a vertex, otherwise if possible an edge.

In all of these problems, the existence and cost of a solution depends on many factors including the input values, the ring size n, the (lack of) agreement on the orientation of the ring, etc. In particular, like any non-trivial problem in anonymous rings [4], they are unsolvable if n is unknown to the processors; hence we assume that n is known.

In this paper we focus on the interrelationship, computability, and complexity of all these problems.
We first provide a characterization of their relationship. Interestingly, we prove that the solvability relationship among these problems depends on the value of $\delta(S)$. As we show, the characterization is rather simple for the cases $\delta(S) \neq 2$; the situation $\delta(S) = 2$ is more complex as it depends on several factors including the ring orientation and the value of n. Some of the obtained characterization results are of particular independent interest. For example, it is well known that, if S is not a set, the election problems are in general not solvable. We provide a characterization by showing that, for a general multiset, GEP is solvable *if and only if* S is aperiodic [3].

We then focus on the complexity of solving these problems for Boolean multisets (i.e., $\delta(S) \leq 2$). We establish an $\Omega(\sum_{j=1}^{l}((z_j)^2+(t_j)^2))$ *lower bound* on the number of messages for solving the sorting and election problems, where z_j and t_j are the lengths of the consecutive blocks of 0's and 1's in S, respectively, and l is the number of such blocks. We then construct an *upper bound* of $O(\sum_{j=1}^{l}((z_j)^2+(t_j)^2)+n\log n)$ for prime n. We do so by presenting an algorithm for oriented rings of prime size, which solves the sorting and election problems using a number of messages bounded as above. These results are easily extended to the unoriented case.

The paper is organized as follows. In Section 2, we describe the framework, define the problems and establish some simple properties of cyclic strings. In Section 3, we examine the relationship between the multiset sorting and election problems. In Section 4, we study the Boolean case, and we establish upper and lower bounds. The Appendix contains a detailed description of the algorithm for oriented rings of prime size.

2. THE FRAMEWORK

2.1. Definitions and Properties

Let $R = r_0 \ldots r_{n-1}$ be an asynchronous ring of n anonymous processors (or *vertices*). That is, each vertex r_i is connected to r_{i-1} and r_{i+1} [4], all vertices are identical, and communication delays (which include, transmission, queuing and processing delays) are finite but unpredictable. We say that R is *oriented* if all processors agree on the same direction (e.g., clockwise), otherwise R is *unoriented*.

To each vertex r_i we associate an input value s_i from a totally ordered set $\mathcal{V}$ of v elements; for simplicity, we assume $\mathcal{V} = Z_v = \{0, \ldots, v-1\}$, but all results hold for an arbitrary totally ordered set. The right (respectively, left) *d-neighbourhood* of r_i is the sequence $\langle s_{i+j} : 0 \leq j < d \rangle$ (respectively, $\langle s_{i-j} : 0 \leq j < d \rangle$).

[3] S is viewed as a circular string.
[4] here and in the following, all operations on the indices are modulo n.

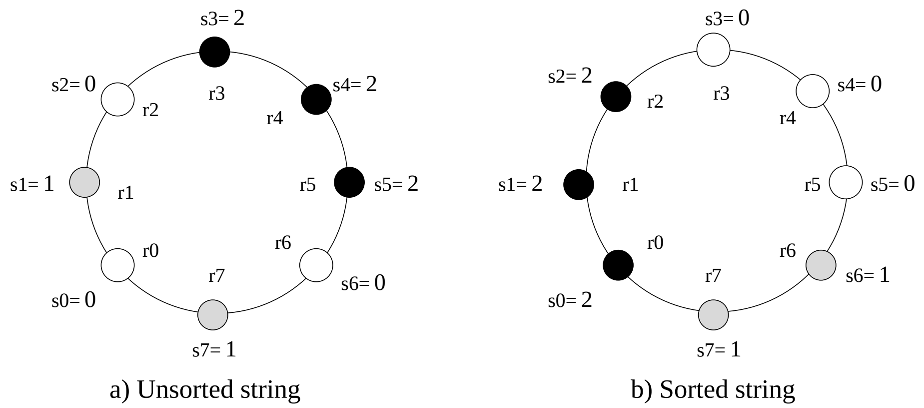

FIG. 1. A a) unsorted and b) sorted ring. Here $\mathcal{V} = Z_3 = \{0, 1, 2\}$, $d_0 = d_2 = 3$, $d_1 = 2$, $\delta(S) = 3$ and $n = 8$.

The values associated with R form the *multiset* $S = \{s_0, \ldots, s_{n-1}\}$ of size n. We denote by $d_u(S)$ the multiplicity of $u \in Z_v$ in S, by $\delta(S)$ the number of distinct values u with $d_u(S) > 0$.

We shall denote by $R(S)$ the ring R that has as input the multiset S. We will consider such a multiset as a (circular) v-*valued string* (or simply *string*) $S = s_0 \ldots s_{n-1}$ of length n. (See Figure 1.)

The string S is *periodic* with *period* k if $S = s_0 \ldots s_{n-1} = (s_0 \ldots s_{k-1})^{\frac{n}{k}}$ and $1 \leq k < n$; otherwise, S is *aperiodic*. We denote by $\bar{S}$ the *reverse* string of S, i.e., $\bar{S} = s_{n-1} \ldots s_0$.

Given $i \in \{0, \ldots, n-1\}$ we denote by $\sigma^i(S)$ the i-th *cyclic shift* (or simply *shift*) of S, i.e., $\sigma^i(S) = s_i s_{i+1} \ldots s_{i-1}$.

The string S is *canonic* if $s_0 \neq s_{n-1}$. Obviously, every S with $\delta(S) > 1$ has a shift that is canonic; therefore, w.l.o.g., we only consider canonic strings where $\delta(S) > 1$.

We denote by $\sigma(S)$ the *multiset* $\sigma(S) = \{\sigma^j(S) | 0 \leq j \leq n-1\}$ and by $\mu(\sigma^j(S))$ the *multiplicity* of $\sigma^j(S)$ in $\sigma(S)$. If $\exists i, 0 \leq i \leq n-1$, such that $\mu(\sigma^i(S)) = 1$ then we say that S is *unique*.

PROPERTY 2.1. *A string S is unique iff it is aperiodic.*

Proof. The property follows from the fact that periodic strings are invariant under a cyclic shift of the size of the period; hence, their multiplicity in $\sigma(S)$ is greater than one. On the other hand, any aperiodic string is trivially unique. ■

A shift $\sigma^i(S)$ of S is *lexicographical minimal* if $\forall j, 0 \leq j \leq n-1$, $\sigma^i(S) \preceq \sigma^j(S)$, where $\preceq$ is the comparison operator between numbers in base v.

PROPERTY 2.2. *In every aperiodic string S the lexicographical minimal shift is unique.*

Proof. Assume by contradiction that there exist at least two lexicographical minimal shifts of S, $\sigma^i(S)$ and $\sigma^j(S)$, with $\sigma^i(S) = s_i \dots s_{i-1} = \sigma^j(S) = s_j \dots s_{j-1}$. By overlapping $\sigma^i(S)$ and $\sigma^j(S)$ it trivially follows that S is periodic, therefore, by property 2.1, we have a contradiction. ■

PROPERTY 2.3. *If n is prime then every string $S \setminus \{0^n, 1^n, \dots, (v-1)^n\}$, is aperiodic.*

Proof. Observe that if a string has period $1 < k < n$, i.e., $S = (x^k)^{n/k}$ then both k and n/k divide n, therefore n is not prime. ■

A v-valued string $S = s_0 \dots s_{n-1}$ is *sorted* iff $\exists\, \sigma^i(S), 0 \leq i \leq n-1$, such that $\sigma^i(S) = 0^{d_0(S)} 1^{d_1(S)} \dots u^{d_u(S)} \dots (v-1)^{d_{v-1}(S)}$, with $d_j(S) \geq 0$, $0 \leq j \leq v-1$, and u^0 is the empty string. Note that for $\delta(S) \leq 2$, if S is sorted, so is $\bar{S}$.

2.2. Problems

We consider several inter-related problems.

PROBLEM 2.1. **Multiset Sorting Problem (MSP)**
Given an (un)oriented ring R and a v-valued string S, move from $R(S)$ to a final configuration $R(S')$ where:

1. $\forall u \in Z_v, \delta_u(S) = \delta_u(S')$;
2. $R(S')$ *is sorted.*

An example for $v = 3$ is shown in Figure 1.

We will study MSP in relation to the classical problems of vertex election and edge election. Following [13] (with a slight adaptation to our case) we define the following:

PROBLEM 2.2. **Vertex Election Problem (VEP)**
Given an (un)oriented ring R with input configuration S, if possible elect a vertex (processor) x as a unique leader, i.e., x knows it has been elected and all the other vertices know they have not been elected.

PROBLEM 2.3. **Edge Election Problem (EEP)**
Given an (un)oriented ring R with input configuration S, if possible elect an edge $e = (x, y)$ as a unique leader, i.e., x and y know which one is e among their incident edges, and all the other vertices know that e is not incident to them.

Vertex election is one of the most basic problems in distributed computing (see [9]); the edge election problem has been studied in detail for anonymous networks in [13]; when an edge is elected and both of its incident vertices know it, they enter a special state.

In addition, we will focus on the more general formulation of the problem which integrates both VEP and EEP.

PROBLEM 2.4. **General Election Problem (GEP)**
Given an (un)oriented ring R with input configuration S, elect a vertex if possible. If a vertex cannot be elected, then elect an edge if possible.

Given a problem P we shall denote by $P(S)$ the instance of P where the input string is S. Given two problems P and Q, we denote by $P \geq Q$ the fact that if a solution σ for P exists, then a solution for Q can be derived from σ. (A similar definition was given in [11].) We denote by $P \equiv Q$ the fact that both $P \geq Q$ and $Q \geq P$ hold.

In the following, when considering upper bounds on the message complexity, we will omit the fact that messages contain at most $O(\log n)$ bits.

3. CHARACTERIZATION

In this section we establish general properties on the solvability of the election and sorting problems, as well as on their relationship. In particular we show that the nature of this relationship depends directly on the value of $\delta(S)$.

3.1. Basic Results

A well known result from [4] states that no non-constant function can be computed on an asynchronous ring if n is not known. Hence in the following we will always assume that *n is known.*

We now establish a necessary and sufficient condition for solvability of GEP. Indeed the following theorem is somehow related to several results of [13]. Unlike in our case, however, such results apply to distinct input values only. We then prefer to state our result as a unique new theorem.

THEOREM 3.1. *Let S be a string given as input to an anonymous ring. $GEP(S)$ is solvable iff S is aperiodic.*

Proof. First consider the case where $GEP(S)$ is solvable, and assume by contradiction that the string is periodic, i.e., $S = (x^k)^{n/k}$ for some x and k. Note that, in this case, vertices at distance k are in the same initial state. W.l.o.g., consider the case of the oriented ring. (If no solution exists for this case, none will exist for the unoriented case.) For any deterministic GEP solution algorithm, consider a synchronous execution on S; in any such execution, at each step, vertices at distance k receive the same values, execute the same operations and, thus, move to the same state. This implies that, in the case of vertex election, if GEP is solvable then $\frac{n}{k}$ vertices will be elected; similarly in the case of edge election, solvability of GEP implies election of $\frac{n}{k}$ edges, a contradiction.

Let us assume S is an aperiodic string and let us show how to solve GEP. The oriented case is trivial, since every aperiodic string S in an oriented ring R has a unique minimal lexicographical shift (property 2.2): the vertex which has the first value of such a string can be elected. This string can be determined at each vertex by performing an input collection algorithm, i.e., by sending all values around the ring, allowing all processors to collect all values.

If the ring is unoriented, this process cannot be applied. Every processor can however perform input collection in both directions and determine the lexicographical minimal shift in each direction. If these two strings are different, the processor that has as input the first value in the smallest of the two becomes the leader. If these strings are not different and the same processor has the first value of both, it becomes the leader. Finally, consider the case when the strings are the same but two distinct processors, x and y, have the first values. Consider the two paths connecting x and y in R. We shall distinguish several cases depending on whether the two paths are even or odd (i.e., they contain an even or odd number of processors, respectively). Let only one be odd; then a leader can be elected (e.g., the middle vertex). A leader can be elected also if both are odd: if the paths have different lengths, the vertex in the middle of the shorter path is elected; otherwise, the substrings associated to the two paths are compared and the vertex in the middle of the lexicographical minimal substring is chosen (the substrings have trivially to be different because the string S is aperiodic).

If they are both even, an edge can be elected (e.g., the one in the middle of the shorter path or with the lexicographical minimal value as before). In this case however, to complete the proof, we have to show that a vertex can not be elected. Let $x = s_i$ and $y = s_j$; since they both have the the first value of the same string (in opposite direction, otherwise trivially S is periodic), we have $s_{i+k} = s_{j-k}$ for all k. Hence, for all k, $\sigma^{i+k}(S) = s_{i+k}s_{i+k+1}\dots s_{i+k-1} = s_{j-k}s_{j-k-1}\dots s_{j-k+1} = \sigma^{j-k}(\bar{S})$. This implies that any deterministic algorithm has a synchronous execution in which the pairs s_{i+k} and s_{j-k} start in the same initial state, and at each step receive the same values, execute the same operations and thus move to the same state. Hence no single vertex can be elected. ∎

We now state the following obvious result:

LEMMA 3.1. *$VEP \geq EEP$.*

Another simple lemma is the following:

LEMMA 3.2. *$VEP \geq MSP$.*

Proof. The leader chooses an arbitrary direction, computes $d_u(S)$ (e.g., by sending counters around the ring) for each $u \in Z_v$, and communicates to every other vertex both its distance from it and the ordered sequence of $d_u(S)$. Based on this information, every vertex can then unambiguously determine its value in the sorted sequence starting from x, and change its value accordingly. ∎

3.2. The simple cases: $\delta(S) \neq 2$

Consider first the case $\delta(S) = 1$.

THEOREM 3.2. *If $\delta(S) = 1$ then GEP is unsolvable, MSP is already solved.*

Proof. If $\delta(S) = 1$ then the system is anonymous and neither a vertex nor an edge can be elected as a leader [1]; thus, GEP is unsolvable. On the other hand the string is by definition sorted, since it consists of a single value. ■

It is interesting to observe that to recognize whether $\delta(S) = 1$ is an expensive process. It is in fact equivalent to the problem of computing the AND of a Boolean string (i.e., the function that is 1 if and only if all inputs are 1) which requires $\Omega(n^2)$ messages [4].

Another simple case is $\delta(S) > 2$.

THEOREM 3.3. *If $\delta(S) > 2$ then $MSP(S) \equiv VEP(S)$.*

Proof. By lemma 3.2, $VEP(S) \geq MSP(S)$. We now show that $MSP(S) \geq VEP(S)$. Let S be sorted; since $\delta(S) > 2$, there are at least three distinct values; each vertex can run an input collection algorithm, thus finding out that S is sorted in a given direction. The unique vertex holding the smallest value in S, and having a neighbour with the biggest value, elects itself a leader. ■

3.3. The last case: $\delta(S) = 2$

The only case left is when $\delta(S) = 2$. Unlike the others, this case is rather complex; we will be using several technical lemmas. In the following, w.l.o.g., we will assume that the two values in the sequence are 0 and 1.

LEMMA 3.3. *$EEP \geq MSP$.*

Proof. Assume an edge has been elected. Let e be the elected edge, and let x and y be the incident vertices. In the absence of an orientation, $d_0(S)$ is computed (redundantly) in both directions; two cases arise depending on whether $d_0(S)$ is even or odd. If $d_0(S)$ is even, the first $d_0(S)/2$ vertices on both sides of e (including x and y) become 0, all others become 1. The case $d_0(S)$ odd is more complex. If n is even, the strings starting with x and then y in one direction and with y and x in the other direction, and ending with edge e are distinct (and can be computed by x and y by doing input collection in both directions), hence a leader can be uniquely chosen. If n is odd, a leader is uniquely determined (e.g., the only vertex at distance $(n-1)/2$ from both x and y). In both cases the chosen leader computes $d_0(S)$ (e.g., by sending a counter around the ring) and tells the closest $d_0(S)$ vertices in an arbitrary direction to assume value 0, and the remaining vertices to assume value 1. In the oriented case, one of the two extremes of e becomes the leader, then computes $d_0(S)$ and sorts as above. ■

THEOREM 3.4. *In oriented rings, $VEP \equiv MSP$.*

Proof. By lemma 3.1 and lemma 3.3, $VEP \geq MSP$. Let S be sorted; then the vertex having the first 0 in the given orientation is uniquely determined and can be elected. ■

In the case of unoriented rings, the relationship between these problems is slightly more complicated.

LEMMA 3.4. *In unoriented rings*

- *if $d_0(S)$ is odd, then $MSP(S) \geq VEP(S)$;*
- *if n is odd, then $MSP \geq VEP$.*

Proof. Assume S is sorted; i.e., $S = s_0 \dots s_{n-1} = 0^{d_0(S)}1^{d_1(S)}$. If $d_0(S)$ is odd, then the vertex $r_{\lfloor (d_0(S)/2) \rfloor}$ is uniquely determined by input collection, and is elected as a leader. If n is odd and $d_0(S)$ is odd we are back to the previous case, otherwise $d_1(S)$ must be odd and the vertex $r_{d_0(S)+\lfloor d_1(S)/2 \rfloor}$ is then uniquely determined by input collection, and thus becomes a leader. ■

LEMMA 3.5. *In unoriented rings, if both n and $d_0(S)$ are even, then $MSP(S) \geq EEP(S)$.*

Proof. Assume S is sorted; i.e., $S = s_0 \dots s_{n-1} = 0^{d_0(S)}1^{d_1(S)}$. A unique edge can be determined and thus elected, e.g., the edge incident on vertices $r_{(d_0(S)/2)-1}$ and $r_{(d_0(S)/2)}$. ■

THEOREM 3.5. *In unoriented rings, $MSP \equiv EEP$.*

Proof. By lemma 3.3, we have $EEP \geq MSP$. To complete the proof we must only show that $MSP(S) \geq EEP(S)$, for every S. Given an arbitrary string S, if both n and $d_0(S)$ are even then by lemma 3.5, $MSP(S) \geq EEP(S)$; otherwise by lemma 3.1 and lemma 3.4 we have $MSP(S) \geq VEP(S) \geq EEP(S)$. Thus $MSP \geq EEP$, completing the proof. ■

LEMMA 3.6. *In unoriented rings, if either n or $d_0(S)$ is odd, then $VEP(S) \equiv MSP(S)$.*

Proof. By lemma 3.1 and lemma 3.3, $VEP \geq MSP$. By lemma 3.4, if either n or $d_0(S)$ is odd then $MSP(S) \geq VEP(S)$. ■

LEMMA 3.7. *In unoriented rings, if $VEP(S) \equiv MSP(S)$, then either n or $d_0(S)$ is odd.*

Proof. Let both n and $d_0(S)$ be even. We will show that $MSP(S) \neq VEP(S)$. Assume S is sorted; e.g., $S = s_0 \dots s_{n-1} = 0^{d_0(S)}1^{d_1(S)}$, and let $j = d_0(S) - 1$. Since the ring is unoriented, r_0 and r_j can not be uniquely distinguished from each other. At the same time, $s_k = s_{j-k}$ for all k; hence, $\sigma^k(S) = s_k s_{k+1} \dots s_{k-1} = s_{j-k}s_{j-k-1} \dots s_{j-k+1} = \sigma^{j-k}(\bar{S})$. In other words r_k and r_{j-k} cannot be distinguished. This implies that any deterministic algorithm has a synchronous execution in which r_k and r_{j-k} start in the same initial state, at every step perform the same operations and receive the same value; thus they move to the same state and remain indistinguishable. Therefore no unique leader can be elected. ■

By lemma 3.6 and lemma 3.7, it follows:

THEOREM 3.6. *In unoriented rings* $VEP(S) \equiv MSP(S)$ *if and only if either* n *or* $d_0(S)$ *is odd.*

By the definition of GEP, from theorem 3.5 and theorem 3.6, it follows that:

THEOREM 3.7. *In unoriented rings,* $GEP \equiv MSP$*.*

Certain values of n can ensure that a solution to GEP and MSP exists. By property 2.3, theorem 3.1, theorem 3.4, theorem 3.6, and theorem 3.7 we immediately have:

THEOREM 3.8. *If* n *is prime then* MSP *and* VEP *are solvable.*

4. BOUNDS ON THE BOOLEAN CASE

In this section we study the Boolean case ($\delta(S) \leq 2$) and establish bounds in the case of prime n. The results apply both to oriented and unoriented rings. We will present them in detail for the oriented case, and describe how to extend them to the unoriented case.

4.1. Lower Bounds

Any string S can be viewed as a sequence of pairs of alternating blocks of 0's and 1's whose lengths are denoted by z_j^S and t_j^S, respectively. Let l^S denote the number of such pairs. Where no ambiguity arises we will omit the superscript. The XOR function is defined by $XOR(S) = d_1(S) \bmod 2$, where $d_1(S)$ ($d_0(S)$) is the multiplicity of 1 (0) in S.

LEMMA 4.1. *Any algorithm which correctly computes XOR on all inputs, requires at least* $\Omega(\sum_{j=1}^{l}((z_j)^2 + (t_j)^2))$ *messages, on input* S*.*

Proof. W.l.o.g., we consider the oriented case, since the bound for the unoriented case follows immediately. Using a mechanism similar to the one introduced in [4] we use as an adversary a synchronizing scheduler that keeps computations as symmetric as possible and delivers messages in *cycles* but delays messages across the boundary of each block. Note that messages have to be exchanged, since otherwise processors would incorrectly compute the XOR function only based on their input value. The computation proceeds as follows: Every processor starts at cycle one and at a generic cycle i receives left and right messages sent at cycle $i-1$, executes some actions and sends new messages; its state therefore depends only on its left and right i-neighbourhoods. Let us first assume $S \notin \{0^n, 1^n\}$, and, w.l.o.g., consider a single block b_j of z_j 0's, and the middle $\lfloor \frac{z_j}{3} \rfloor$-processors of b_j (see Figure 2).

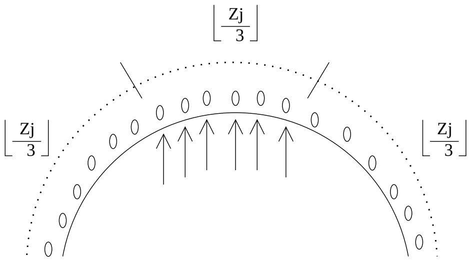

FIG. 2. Block b_j of z_j 0's.

The left and right $\lfloor \frac{z_j}{3} \rfloor$-neighbourhood of all such processors is composed of all 0's, and therefore for at least $\lfloor \frac{z_j}{3} \rfloor$ cycles the $\lfloor \frac{z_j}{3} \rfloor$ processors will move to the same state.

Observe that, if these processors compute the XOR function at cycle $t < \lfloor \frac{z_j}{3} \rfloor + 1$ (i.e., before observing a different bit), then the adversary can choose to complete the string S with $n-t$ bits so that the output of $XOR(S)$ is different from the one computed by at least one of these processors. (Note that $XOR(S) \neq XOR(S')$ if S and S' differ in a singe bit.) This implies that at least $\lfloor \frac{z_j}{3} \rfloor$ cycles must pass and at least $\lfloor \frac{z_j}{3} \rfloor \lfloor \frac{z_j}{3} \rfloor$ messages will have been sent before one of the processors in the block observes a different value and moves to a different state.

Obviously, this holds for every block of length z_j and therefore globally at least $\Omega(\sum_{j=1}^{l} (z_j)^2)$ messages must be sent. A similar argument can be provided for the blocks of 1's.

For the case of $S = 0^n$ ($S = 1^n$, respectively) we can use the $\Omega(n^2)$ messages lower bound of [4]. Note that however $z_1 = n$ ($t_1 = n$, respectively), and therefore the bound of the lemma follows. ∎

We now establish a basic relationship between MSP and the problem of computing the XOR of a string S with $\delta(S) = 2$.

LEMMA 4.2. *$MSP \geq XOR$ using a reduction that requires $O(n)$ messages.*

Proof. First observe that the XOR function is invariant with respect to orientation (i.e., $XOR(S) = XOR(\bar{S})$). Assume S is sorted, i.e., $S = 0^{d_0(S)}1^{d_1(S)}$. Starting from s_0 (and also $s_{d_0(S)-1}$ if the ring is unoriented) compute $d_1(S)$ (and $d_1(\bar{S})$ if unoriented) by sending a counter (two counters) around the ring; then $XOR(S) = XOR(\bar{S}) = d_1(S) \bmod 2 = d_1(\bar{S}) \bmod 2$ is easily computed and broadcasted to all processors using a further n messages. ∎

From the above we derive the following:

THEOREM 4.1. *Given a string S, the problems $GEP(S)$ and $MSP(S)$ require at least $\Omega(\sum_{j=1}^{l}((z_j)^2 + (t_j)^2))$ messages in an asynchronous ring.*

Proof. From lemma 4.2 we know that a lower bound for the XOR function is a lower bound for the $MSP(S)$ (to within an additive factor of $O(n)$) and from theorem 3.7 the same holds for $GEP(S)$. The result follows by lemma 4.1. ■

4.2. Upper Bounds

In this section, we present two algorithms one for oriented and one for unoriented rings of n processors, in the case of prime n. The two algorithms exchange at most $O(\sum_{j=1}^{l} ((z_j)^2 + (t_j)^2 + n\log n))$ messages and solve $GEP(S)$ or $MSP(S)$, if a solution exists; in the case no solution exists, all vertices become aware of this fact.

4.2.1. Oriented Ring

In this section we consider an *oriented* ring of n processors, with prime n, and we present an algorithm (run by each processor, e.g., by p) for solving the leader election and sorting problems.

The general idea is to assign to each active processor dynamically changing labels and to decrease step by step the number of active processors (by comparing neighbouring labels), up to a state in which only one is active and may elect itself a leader, tell the other processors they are defeated and eventually sort (if required).

Formally, a processor p starts as active in an *Initial* state with an input bit b; it sends this bit to the right and moves to a *SeenOnlyEqual* state. Intuitively, p remains in this new state as long as it "sees" on its left only processors with the same input b. The number of such processors will eventually determine its new label that will be used in the next state. More precisely, many cases may arise: a) p receives a total of $n-1$ bits equal to b from the left and in this case it moves to an *All-Equal* state since it detects that $S \in \{0^n, 1^n\}$ and therefore the algorithm can end (no leader can be elected and S is already sorted); b) p sends bits to the right and receives bits from the left until it receives a bit $\neq b$; it chooses as a new label v the number of b's it has collected from the left plus its own, sends this value to the right and to the left in a $< SOE, v >$ message, and then moves to an *Electing* state; c) it receives a $< SOE, z >$ message from a neighbouring processor. It stores the new label z of its neighbour; moreover if $< SOE, z >$ comes from the left it chooses as its new label the value $z+1$ (as this message is equivalent to the reception of a bit $\neq b$) and then moves to the Electing state.

Intuitively in the Electing state the number of active processors has to decrease. Formally, a processor p first receives (unless this was done in the previous state) the $< SOE, x >$ and $< SOE, y >$ messages, containing the new labels (values x and y) of its active left and right neighbours. It then compares its value v with x and y. If v is such that $x \leq v$ and $y < v$ or $x < v$ and $y \leq v$, then p remains active, otherwise it becomes passive and moves to a *Passive* state. If p remains active, it then sends a counter (initialized to 1) to the right and moves to a *Counting* state. In this state the remaining active processors update their labels into a new one computed as follows. Passive processors that receive the counter increase it of 1 and forward it to the right. An active processor p receiving a counter d from the left checks if $d = n$. In this case p knows that this is its own counter as all the other $n-1$ processors are passive. It therefore becomes a leader and moves to an *Elected* state. Otherwise, $d < n$ and p chooses the value d as its new label, sends a $< SOE, d >$ message to the left and to the right and moves back to the Electing state. As the number of active processors decrease at each iteration of the Electing and Counting states, at a certain point there will be a unique leader that moves to the Elected state. In this state the leader has to complete respectively the election or the sorting problem. Formally, in case of election, the leader sends its value around and enters a final state *Leader.* While receiving this message all other processors move from a Passive to a *Defeated* state (where they know they have not been elected) and stop. In case of sorting, the leader determines (by circulating a counter) $d_0(S)$, chooses value 0 and by circulating a message tells the first $d_0(S)-1$ processors on its right to change their input bit into 0 and the others into 1; once a processor knows its final value it moves to a *Sorted* state and stops.

A formal description of the algorithm is contained in the Appendix. We assume, without loss of generality, that all processors start independently the execution of the algorithm. (Should this not be the case, a "wake-up" preprocessing phase requiring $O(n)$ messages can be added.)

THEOREM 4.2. *The above algorithm correctly solves the $GEP(S)$ and $MSP(S)$ for prime n (if $S \in \{0^n, 1^n\}$ it reports $GEP(S)$ is unsolvable); it exchanges at most $O(\sum_{j=1}^{l}((z_j)^2+(t_j)^2)+n\log n)$ messages in the worst case.*

Proof. Let us first prove the correctness. In the SeenAllEqual state there are two cases: 1) all processors detect that $S \in \{0^n, 1^n\}$ and stop as no leader can be elected and the string is already sorted; 2) at least one processor detects that $S \notin \{0^n, 1^n\}$ and moves to an Electing state.

Note that, if $S \notin \{0^n, 1^n\}$, as n is prime, then S is aperiodic (see property 2.3). This implies that in the first Electing state at least one of the processors has a value different from one of its neighbours, i.e., values are not all equal. The same holds also in the next Electing states, during which labels are distances between neighbouring active processors (the sum of all such distances gives n, but n is prime therefore they cannot be all equal).

We want now to prove that in every Electing state at least one processor remains active. A processor with maximal value, which has a neighbouring active processor with a smaller value will remain active. Such processor does not exist only if values are all equal but this is not possible for what we have proved above.

Note now that, in every Electing state at least $1/3$ of the processors become passive since a processor remains active if it has a value larger than at least one of its neighbours (that in this case becomes passive). This implies that both this and the Counting states are repeated at most $\log_{1.5} n = O(\log n)$ times.

In the Elected state a unique processor is active and it can trivially elect itself as a leader, count $d_0(S)$ (by sending a counter around the ring) and sort the string.

For the complexity cost, observe that in the SeenOnlyEqual state every processor collects at most a set z_j or t_j of bits, depending on the block it receives. More precisely every bit 0 (1) in block z_j (t_j) at distance $i \le z_j$ from the first bit 1 (0) travels at most i steps, therefore in block z_j at most $\sum_{i=1}^{z_j} i = \frac{z_j(z_j+1)}{2}$ ($\sum_{i=1}^{t_j} i = \frac{t_j(t_j+1)}{2}$) bits travel. In total at most $\sum_{j=1}^{l}(\frac{z_j(z_j+1)}{2}+\frac{t_j(t_j+1)}{2})$ messages travel. Moreover in the case of $S \in \{0^n, 1^n\}$, $O(n^2)$ messages travel (and $z_1 = n$ or $t_1 = n$). Note that only $O(n)$ messages are necessary for counters. At the Electing state at least

$1/3$ of the processors become passive in each round, and there are at most $O(\log n)$ rounds during which each active processor sends a counter. The total number of messages exchanged in this state is then $O(n \log n)$. Finally at the Elected state the leader computes $d_0(S)$ and sorts using a counter (i.e., sending twice at most $O(n)$ messages). The total number of messages exchanged is therefore at most $O(\sum_{j=1}^{l}((z_j)^2 + (t_j)^2) + n \log n)$. ■

4.2.2. Unoriented Ring

In the case of an *unoriented* ring, every processor will be involved into two separate executions, one for each direction, of the algorithm for the oriented case. Two situations are possible as a result of the two independent executions. If $\delta(S) = 1$ every processor knows that $S \in \{0^n, 1^n\}$ and therefore that $MSP(S)$ is already solved and $GEP(S)$ is unsolvable. If $\delta(S) = 2$, then two leaders are elected; the two leaders, x and y, will then send a counter in both directions, in order to compute the two distances (i.e., the number of processors inside the path) to the other leader. Note that n is prime, therefore one of the two distances is odd (and the other is even). The leaders x and y can then send an election message to the processor in the middle of the odd path. This processor moves to a *NewElected* state and can eventually sort. As usual, passive processors in the Passive state forward (and, if appropriate, increase) the received values. Observe that, since the two executions of the algorithm for the oriented case are run concurrently and independently, a processor can be in different states with respect to each execution. Note that it is possible that a passive processor becomes elected (because it is in the middle of the path). We now have:

THEOREM 4.3. *For the case prime n, $GEP(S)$ and $MSP(S)$ can be solved in an anonymous unoriented ring optimally exchanging at most $O(\sum_{j=1}^{l}((z_j)^2 + (t_j)^2) + n \log n)$ messages in the worst case.*

Note that the upper bound of theorems 4.2 and 4.3 differs from the lower bound of theorem 4.1 for an $n \log n$ additive term. Since the lower bound can be as low as $\Omega(n)$ (when z_i and t_i are $\Theta(1)$ for all i), the asymptotic difference between the two bounds is always reasonably small. The two bounds may match, e.g., when for a given i, z_i or t_i is of order $\Theta(n)$.

5. CONCLUSIONS

In this paper we have considered the problem of distributively sorting a multiset of input values in a ring of n processors. We have studied its properties and we have investigated its relationship with the leader election problem. We have also studied the communication complexity of the above problems for Boolean S and prime n, establishing two reasonably close lower and upper bounds.

There are several obvious extensions, among them the establishment of an upper-bound for the non-Boolean case and tight bounds for the Boolean case.

It would be interesting to determine if an analogous relationship between sorting and leader election exists in networks with other symmetric topologies (e.g., hypercube, torus, etc). A negative answer would be particularly intriguing.

ACKNOWLEDGMENT

This work has been supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada), MURST (progetto 'Matematica per le scienze e la tecnologia", Università di Trieste), MITACS (Federal Center of Excellence on Mathematics of Information Technology and Complex Systems), Nortel Networks.

REFERENCES

1. D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12-th ACM Symposium on Theory of Computing*, pages 82–93, 1980.
2. H. Attiya and Y. Mansour. Language complexity on the synchronous anonymous ring. *Theoretical Computer Science*, 53:169–185, 1987.
3. H. Attiya and M. Snir. Better computing on the anonymous ring. *Journal of Algorithms*, 12(2):204–238, 1991.
4. H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–875, 1988.
5. H.L. Bodlaender, S. Moran, and M.K. Warmuth. The distributed bit complexity of the ring: from the anonymous to the non-anonymous case. *Information and Computation*, 108(1):34–50, 1994.
6. P. Ferragina, A. Monti, and A. Roncato. Trade-off between computation power and common knowledge in anonymous rings. In *Proceedings of the 1st Colloquium on Structural Information and Communication Complexity*, pages 35–48, 1994.
7. P. Flocchini, E. Kranakis, D. Krizanc, F.L. Luccio, and N. Santoro. Sorting multisets in anonymous rings. In *Proceedings of the 14-th IEEE International Parallel and Distributed Processing Symposium*, pages 275–280, 2000.
8. E. Kranakis, D. Krizanc, and F.L. Luccio. On recognizing a string on an anonymous ring. *Theory of Computing Systems*, 34(1):3–12, 2001.
9. N. Lynch. *Distributed Algorithms.* Morgan-Kaufmann, San Mateo, CA, 1996.
10. S. Moran and M.K. Warmuth. Gap theorems for distributed computation. *SIAM Journal on Computing*, 22(2):379–394, 1993.
11. N. Sakamoto. Comparison of initial conditions for distributed algorithms on anonymous networks. In *Proceedings of the 18th Annual ACM Symposium on Distributed Computing, Atlanta, Georgia, USA*, pages 173–179, 1999.
12. M. Yamashita and T. Kameda. Electing a leader when processor identity numbers are not distinct. In *Proceedings of the 3rd International Workshop on Distributed Algorithms, Nice, France*, pages 303–314, 1989.
13. M. Yamashita and T. Kameda. Computing on anonymous networks, part I: Characterizing the solvable cases. *IEEE Transaction on Parallel and Distributed Systems*, 7(1):69–89, 1996.
14. M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transaction on Parallel and Distributed Systems*, 9(10):878–887, 1999.

APPENDIX

Algorithm for oriented rings

The algorithm is specified as a set of rules, ($state \times event \rightarrow action$). By default, absence of "action" denotes the null action.

At each processor the algorithm uses the following local variables.
b - input and final value of the processor;
$I, SEO, E.1, E.2, SOR.1, SOR.2$ - header of the messages sent in different states;
$c, d, z, l, counter$ - counters;
$v, value, x, y, newy$ - processor labels;
w - Boolean value.

Set of states:
INITIAL - initial state;
ALLEQUAL - stop since $S \in \{0^n, 1^n\}$;
SEENALLEQUAL - detect if $S \in \{0^n, 1^n\}$ or get a new label;
ELECTING - be selected or become passive;
COUNTING - become a leader or be selected again;
ELECTED - send election or sorting message around;
SORTING - compute number of 0's;
PASSIVE - forward and eventually change messages and become sorted or defeated;
SORTED,*LEADER*,*DEFEATED* - terminal states.

Set of rules (see also Fig. 1 of the appendix):
INITIAL
```
    x, y, z, l, newy := ∅;
    c := 1;
    send ⟨I, b⟩ to right;
    become SEENALLEQUAL;
```

SEENALLEQUAL
```
    receiving ⟨I, w⟩ from left do
        if w ≠ b then v := c;
                      send ⟨SOE, v⟩ to both directions;
                      become ELECTING
                 else c := c + 1;
                      if  c = n then become ALLEQUAL
                             else send ⟨I, b⟩ to right
                      fi
        fi;
    receiving ⟨SOE, value⟩ from left do
        y := value; /* this is the new label of my left neighbour */
        l := 1;
        v := value + 1; /* this is my new label */
        send ⟨SOE, v⟩ to both directions;
        become ELECTING;
```

```
    receiving ⟨SOE, value⟩ from right do
        x := value; /* this is the new label of my right neighbour */

ALLEQUAL
    stop;

ELECTING
    while (x = ∅) or (y = ∅) do /* I wait for the new labels of my neighbours */
        receiving ⟨I, value⟩ from left do remove it; /* old message */
        receiving ⟨SOE, value⟩ from right do x := value;
        receiving ⟨SOE, value⟩ from left do
                                if l = ∅ then l := 1;
                                              y := value;
                                         else newy := value;
        receiving ⟨E.1, d⟩ from left do z := d;
    if x = y = v or x > v or y > v then if z ≠ ∅ then send ⟨E.1, z + 1⟩ to right;
                                        if newy ≠ ∅ then send ⟨SOE, newy⟩ to right;
                                        become PASSIVE
                                   else send ⟨E.1, d := 1⟩ to right;
                                        become COUNTING
    fi;

COUNTING
    if ((z ≠ ∅) or receiving ⟨E.1, d⟩ from left) then
              if (z ≠ ∅) then d := z;
              if d = n then become ELECTED /* all others are passive */
                   else v := d; /* my new label */
                        x := ∅;
                        if newy ≠ ∅ then y := newy;
                                         l := 1;
                                         newy := ∅;
                                    else y, l := ∅;
                        send ⟨SOE, v⟩ to both directions;
                        become ELECTING
              fi;

ELECTED
    if Problem.type=Election then send ⟨E.2, value⟩ to right;
                                  become LEADER
                             else /* Problem.type=Sorting */
                                  if b = 0 then counter := 1
                                           else counter := 0;
                                  send ⟨SOR.1, counter⟩ to right;
                                  become SORTING
    fi;

SORTING
```

```
    receiving ⟨SOR.1, counter⟩ from left do
                        b := 0;
                        send ⟨SOR.2, counter − 1⟩ to right;
                        become SORTED

PASSIVE
    receiving ⟨HEAD, value⟩ from direction do
            case HEAD of
                SOE : send ⟨SOE, value⟩ to opposite(direction);
                E.1 : send ⟨E.1, value + 1⟩ to opposite(direction);
                E.2 : send ⟨E.2, value⟩ to opposite(direction);
                      become DEFEATED;
                SOR.1 : if b = 0 then value := value + 1;
                      send ⟨SOR.1, value⟩ to opposite(direction);
                SOR.2 : if value > 0 then b := 0 else b := 1;
                      send ⟨SOR.2, value − 1⟩ to right;
                      become SORTED
            end;

SORTED
    stop
LEADER
    stop
DEFEATED
    stop
```

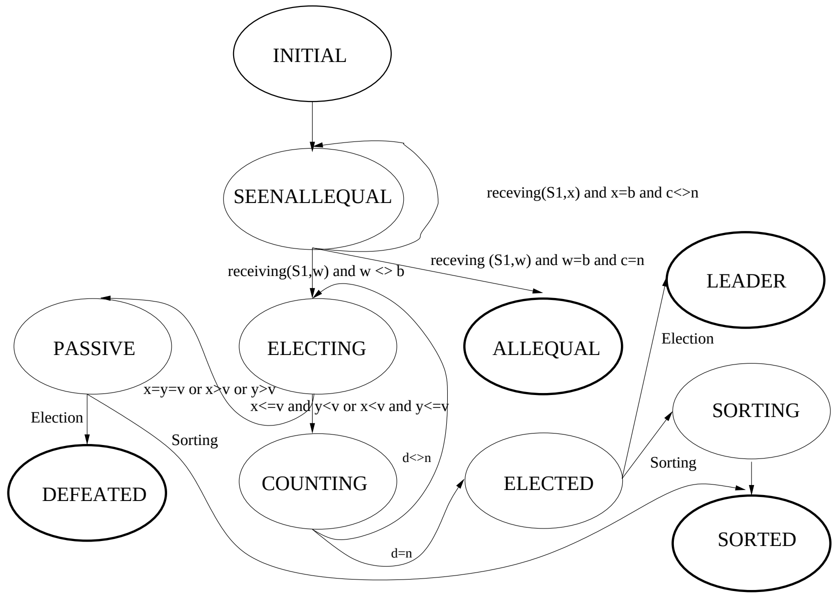

FIG. 1. Set of rules.