Sorting Multisets and Electing a Leader in Anonymous Rings*

P.Flocchini[†] E. Kranakis[‡] D. Krizanc[§] F.L. Luccio[¶]
N. Santoro[†]

DRAFT

Abstract

An anonymous ring network is a ring where all processors (vertices) are totally indistinguishable except for their input value. Initially, to each vertex of the ring is associated a value from a totally ordered set; thus, forming a multiset.

In this paper we consider the problem of sorting such a distributed multiset and we investigate its relationship with the election problem.

We focus on the computability and the complexity of these problems, as well as on their interrelationship, providing strong characterizations, showing lower bounds, and establishing efficient upper bounds.

1 Introduction

An *anonymous* ring network $R = r_1, \ldots r_n$ is a ring where all processors (or vertices) are totally indistinguishable except for their input value. Notice that indices are used only for clarity of discussion and are not known to the vertices. From a computational viewpoint, anonymous systems are the least powerful distributed systems, and hence the ideal setting to study the complexity of problems and the relationship among them.

*This work has been supported in part by N.S.E.R.C. and M.U.R.S.T., progetto 40 % "Metodi e Problemi in Analisi Reale".

[†]School of Information Technology and Engineering, University of Ottawa, Canada. flocchin@-site.uottawa.ca

[‡]School of Computer Science, Carleton University, K1S 5B6 Ottawa, Canada. {kranakis, santoro}@scs.carleton.ca

[§]Mathematics Department, Wesleyan University, Middletown, CT, 06459, U.S.A.. krizanc@-wesleyan.edu

[¶]Dipartimento di Scienze Matematiche, Università degli Studi di Trieste, Trieste, Italy. luccio@mathsun1.univ.trieste.it

Initially, to each vertex r_i of the ring is associated a value s_i from a totally ordered set $\mathcal{V}$. The associated values form a *multiset* $S = \{s_1, \ldots, s_n\}$, where $s_i \in \mathcal{V}$. Let us denote by $\delta(S)$ the number of distinct elements of $\mathcal{V}$ which appear in S.

The particular case when $\delta(S) = n$ 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 really anonymous since the distinct values can be used as identities and allow to distinguish among the vertices. Networks with distinct values 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 in several asynchronous network topologies (e.g., [11]).

In the case of a general multiset (i.e. $\delta(S) < n$), unfortunately very little is known. Most of the existing results focus on the Boolean case (i.e., $\delta(S) \leq 2$) and study the problem of computing Boolean functions (such as OR, AND, XOR) [3, 4, 5, 6, 8, 12]. For the general case, only a few distributed problems have been studied: finding the extrema of the multiset [1], and finding the multiplicity of a value in the multiset [1].

In this paper we consider the problem of *sorting* the multiset (*MSP*) in asynchronous anonymous rings, and we investigate its relationship with the vertex and edge election problems (*VE*, *EE* respectively) and with the general election problem GEP.

Sorting the multiset means that, at the end of the computation, the values must be placed on the ring so to be ordered in a direction starting from a vertex; the choice of the vertex is not predetermined. The sorting problem has been extensively studied when the values are distinct (or the network is not anonymous) (e.g., see [10, 13]); however, it has never been investigated in the case of multisets.

The vertex election problem *VE* consists in starting from a situation where the network is anonymous and ending in a situation where a vertex is different from the others. Analogously, in the edge election problem *EE*, an edge must become different from all others, and identified as the leader. The general election problem *GEP* is the problem of electing, if possible a vertex, otherwise an edge.

Clearly the solvabilty of these problems depends on many factors including the input values, the value of n (the ring size), the fact that the ring is oriented or not, etc. Since any non-trivial problem is unsolvable if n is unknown to the vertices [4], we will assume in the following that n is a priori known.

In this paper we focus on the computability and the complexity of these problems, as well as on their interrelationship, providing strong characterizations, proving lower bounds, and establishing efficient upper bounds. For example, it is well known that, if S is not a set, the election problems are in general not solvable; but no characterization existed. We provide one by showing that, for a general multiset, GEP is solvable if and only if S is aperiodic.

On the relationship among the sorting and the election problems we provide a complete characterization. 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 when $\delta(S) = 2$ is more complex, and it depends on several factors including the ring orientation and the value of n.

We then focus on the complexity of solving these problems and study the Boolean setting (i.e., $\delta(S) \leq 2$). We first present an algorithm for oriented rings of prime size; this algorithm solves the sorting and election problems using at most $O(\sum_{j=1}^{l}({z_j}^2 + {t_j}^2) + n \log n)$ messages, where z_j and t_j are the lengths of the consecutive blocks of 0's and 1's in S, respectively. We also present an algorithm for oriented rings of not-prime size; this algorithm solves the election and sorting problems (if a solution exists) using on average $O(n \log n)$ messages. The same bounds are shown to be achievable also for *unoriented* rings. Finally we present some lower bounds; these bounds prove that the algorithms for the case of n prime are optimal. We also show that, when n is not prime, any solution to these problems requires the exchange of $\Omega(n \log n)$ messages for almost all multisets S; hence, for almost all inputs, the algorithms for the case of n not prime are average-case optimal.

2 Definitions

Let $R = r_1 \dots r_n$ be an anonymous ring of n processors. 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 of the ring it is associated a value s_i from a totally ordered set $Z_v = \{0, \dots, v-1\}$; the associated values form a *multiset* $S = \{s_1, \dots, s_n\}$, where $s_i \in Z_v$.

In the following we will consider such a multiset as a string of values. More precisely, the multiset corresponds to a *v-valued* string $S = s_1 \dots s_n$ of length n where $s_i \in Z_v = \{0, \dots, v-1\}$, $1 \leq i \leq n$. 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 $d_u(S) > 0$, and by $R(S)$ the ring R with input S , i.e., s_i being the input of node r_i.

Given $k \in \{1, \dots, n\}$ we denote by $\sigma^k(S)$ the k-th *cyclic shift* (or simply *shift*) of S, i.e., $\sigma^k(S) = s_k s_{k+1} \dots s_{k-1}$ where all indices are modulo n. We denote by $\sigma(S)$ the *multiset* $\{\sigma^k(S) | 1 \leq k \leq n\}$ and by $\mu(\sigma^k(S))$ the *multiplicity* of $\sigma^k(S)$ in $\sigma(S)$; if $\exists i$ such that $\mu(\sigma^i(S)) = 1$ then we say that S is *unique*. We also say that the string is *periodic* with *period* k if $S = s_1 \dots s_n = (s_1 \dots s_k)^{\frac{n}{k}}$ and $1 \leq k < n$, otherwise the string is *aperiodic*. Finally we denote by $\bar{S}$ the reverse string of S, i.e., $\bar{S} = s_n \dots s_1$.

A v-valued string S is *canonic* if $s_1 \neq s_n$. Obviously, every S with $\delta(S) > 1$ has a shift that is a canonic string; therefore, w.l.o.g., we only consider canonic strings, where $\delta(S) > 1$.

Property 1 *A string S is unique iff it is aperiodic. Furthermore, S is unique iff $\bar{S}$ is unique.*

PROOF The property trivially follows from the fact that periodic cyclic strings are invariant under a shift of the size of the period; hence, their multiplicity in $\sigma(S)$ is

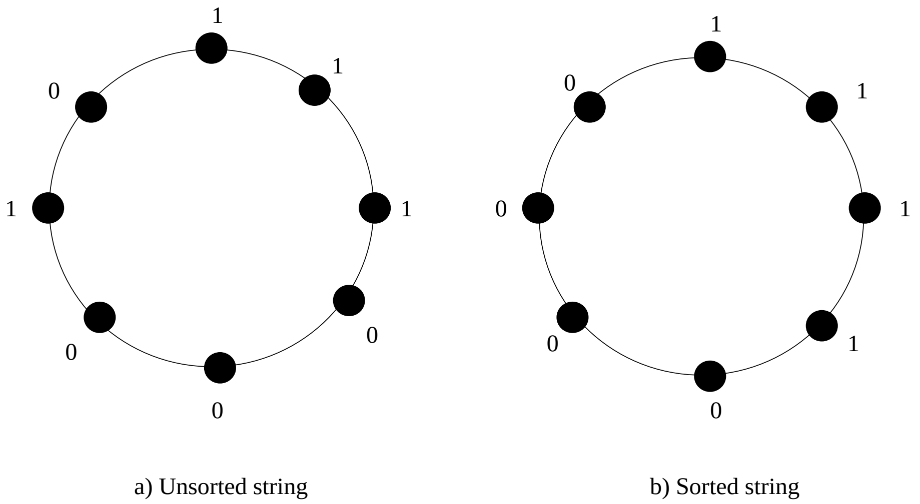

Figure 1: A a) unsorted and b) sorted ring.

greater than one. On the other hand, any aperiodic string is trivially unique. Assume now that S is unique, and thus it is aperiodic, and assume by contradiction that $\bar{S}$ is not unique, i.e., it is periodic with period j. This implies that $\bar{S} = s_n \dots s_1 = (s_n \dots s_r)^j$, hence trivially $S = s_1 \dots s_n = (s_r \dots s_n)^j$, i.e., S is periodic, therefore a contradiction. The other implication can be proved in the same way. ■

Property 2 *In every aperiodic string S the lexicographical minimal shift is unique.*

PROOF Assume by contradiction that there exist at least two lexicographical minimum strings $\sigma^i(S)$ 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 a contradiction. ■

Property 3 *If n is prime then every string S, is aperiodic.*

PROOF If n is prime, n cannot be decomposed in the product of prime numbers but 1 and n. Observe though 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 $S = s_1 \dots s_n$ is *sorted* iff $\exists\, \sigma^i(S), 1 \le i \le n$, such that $\sigma^i(S) = 0^{d_0(S)} 1^{d_1(S)} \dots u^{d_u(S)} \dots (v-1)^{d_{v-1}(S)}$, where u^0 is the empty string. Note that for $\delta(S) \le 2$, if S is sorted, so is $\bar{S}$.

We consider several inter-related problems. The main focus of this paper is on the sorting problem.

Problem 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 $v = 2$ is shown in Figure 1.

We will study MSP in relation to the classical problems of vertex election (VE) and edge election (EE). Vertex election is the most basic problem in distributed computing (see [11]); the edge election problem has been widely studied in detail in [15]; when an edge is elected, both vertices know about it (e.g., are in a special state).

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

Problem 2 General election problem (GEP)
Given an (un)oriented ring R with input configuration S, if possible elect a vertex, else elect an edge.

In the course of our investigation on the Boolean case, we shall also consider the problem of computing well-known Boolean functions such as AND and XOR.

Given a problem P we shall denote by $P(S)$ the instance of P when the input string is S. Given two problems P and Q, we denote by $P \geq Q$ the fact that any solution to P implies a solution to Q, and by $P \equiv Q$ the fact that both $P \geq Q$ and $Q \geq P$.

In the following, when considering the message complexity, we will omit to say that messages are bounded and contain at most $O(\log n)$ bits.

3 General Properties

In this section we first establish some general properties on resolvability of the election and sorting problem, 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 basic negative well known result is the following and follows from [4]:

Lemma 1 *No non-constant function can be computed in asynchronous rings if n is not known.*

From this we have:

Corollary 1 *The GEP and MSP are deterministically unsolvable 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 resolvability of GEP.

Theorem 1 *Let n be known and 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 period k; notice 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, their state will be the same. This implies that, in case of vertex election, if GEP is solvable then $\frac{n}{k}$ vertices will be elected; similarly in case of edge election, resolvability of GEP implies election of $\frac{n}{k}$ edges. A contradiction.

Let us now consider the case of aperiodic strings 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): the vertex who has the first value of such a string can be elected. This string can be determined at each vertex by performing an input collection.

If the ring is unoriented, this process cannot be applied. Every processor can however perform input collection in both directions and determine the minimal lexicographical string in each direction. If these strings are different, the processor that has as the first value in the smallest of the two becomes the leader. If these strings are not different and the same process 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: e.g., the vertex in the middle of the shortest path (since the string is aperiodic, the paths have different lengths).

If they are both even, an edge can be elected (e.g., the one in the middle of the shortest path); 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), 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 with the same initial value, and at each step receive the same values, execute the same operations and thus their state will be the same. Hence no single vertex can be elected. ■

We now recall the following obvious result:

Lemma 2 *($VE \geq EE$).*

Another simple property is the following:

Theorem 2 *$VE(S) \geq MSP(S)$.*

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 Characterization: The Case $\delta(S) \neq 2$

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

Theorem 3 *If $\delta(S) = 1$ then GEP is unsolvable, MSP is already solved.*

PROOF If $\delta(S) = 1$ the system is anonymous and neither a vertex nor an edge can be elected as a leader [2]; 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 a non-trivial problem. The problem P_u of determining whether $S = u^n$ (where $u \in Z_v = \{0, \ldots, v-1\}$) is in fact equivalent to the problem of computing the AND of a Boolean string.

Theorem 4 *$AND \equiv P_u$.*

PROOF Let us first show that $P_u \geq AND$. Let us assume we have solved P_u. If S is a Boolean string, then $Z_v = \{0, 1\}$, and trivially $P_1(S) = AND(S)$ since if $P_1(S) = 1$ then $S = 1^n$ and therefore $AND(S) = AND(1^n) = 1$, otherwise $P_1(S) = 0$, i.e., $S \neq 1^n$ and $AND(S) = 0$. Let us now show that $AND \geq P_u$. We move from a configuration $S = s_1 \ldots s_n$ to $S' = s'_1 \ldots s'_n$ where if r_i has $s_i = u$ then $s'_i = 1$ else $s'_i = 0$. Trivially $AND(S') = P_u$ since $AND(S') = 1$ iff $S = u^n$. ■

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

Theorem 5 *If $\delta(S) > 2$ then $MSP(S) \equiv VE(S)$.*

PROOF By theorem 2, $VE(S) \geq MSP(S)$. Let us now show that $MSP(S) \geq VE(S)$. Let S be sorted; since $\delta(S) > 2$, there are at least three distinct values; the sorted sequence thus uniquely defines an orientation in the ring (e.g., according to the increasing values). Hence a leader can be elected (e.g., the first vertex with the smallest value). ∎

3.3 Characterization: The 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 a sequence of technical lemmas. In the following, w.l.o.g., we will assume that the two values in the sequence are 0 and 1, and will omit to state that $\delta(S) = 2$.

Lemma 3 *($EE \geq MSP$).*

PROOF Assume an edge has been elected. Let e be the elected edge, and let x and y be the incident vertices. In the oriented case, one of the two vertices, say x, 1) becomes the leader, 2) computes $d_u(S)$ (e.g., by sending a counter around the ring), and 3) tells the first $d_0(S)$ vertices to assume value 0 and the rest to assume value 1. In absence of orientation, $d_u(S)$ 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. Consider now the case when $d_0(S)$ is odd. If n is even, the strings starting from x and y and ending with edge e are distinct, 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 all these cases the chosen leader executes steps 2) and 3) of the oriented case. ∎

Theorem 6 *In oriented rings, $VE \equiv MSP$.*

PROOF By lemma 2 and lemma 3, $VE \geq MSP$. Let S be sorted; than 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 4 *In unoriented rings*

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

Proof Assume a string S is sorted; i.e., $S = s_1 \dots s_n = 0^{d_0(S)}1^{d_1(S)}$.

1) If $d_0(S)$ is odd, the vertex $r_{\lceil (d_0(S)/2) \rceil}$ is uniquely determined and can be elected as a leader.

2) if n is odd either $d_0(S)$ or $d_1(S)$ is odd. The first case is already dealt with; in the second case, the vertex $r_{d_0(S)+\lceil d_1(S)/2 \rceil}$ is uniquely determined and can become leader. ■

Lemma 5 *In unoriented rings, if both n and $d_0(S)$ are even, then $MSP(S) \geq EE(S)$.*

Proof Assume S is sorted; e.g., $S = s_1 \dots s_n = 0^{d_0(S)}1^{d_1(S)}$. A unique edge can be determined and thus elected, e.g., the edge incident on vertices $r_{(d_0(S)/2)}$ and $r_{(d_0(S)/2+1)}$. ■

Theorem 7 *In unoriented rings $MSP \equiv EE$.*

Proof By lemma 3, we have $EE \geq MSP$. To complete the proof we must only show that $MSP(S) \geq EE(S)$ for every S. Given an arbitrary string S, if both n and $d_0(S)$ are even then by lemma 5, $MSP(S) \geq EE(S)$; otherwise by lemma 2 and lemma 4 $MSP(S) \geq VE(S) \geq EE(S)$. Thus $MSP \geq EE$, completing the proof. ■

Lemma 6 *In unoriented rings, if n or $d_0(S)$ is odd then $VE(S) \equiv MSP(S)$.*

Proof By lemma 2 and lemma 3, it follows $VE \geq MSP$. By lemma 4, if either n or $d_0(S)$ is odd then $MSP(S) \geq VE(S)$. ■

Lemma 7 *In unoriented rings, if $VE(S) \equiv MSP(S)$ then n or $d_0(S)$ is odd.*

Proof Let both n and $d_0(S)$ be even. We will show that $MSP(S) \neq VE(S)$. Assume S is sorted; e.g., $S = s_1 \dots s_n = 0^{d_0(S)}1^{d_1(S)}$, and let $j = d_0(S)$. Since the ring is unoriented, r_1 and r_j can not be uniquely distinguished from each other. At the same time, $s_{1+k} = s_{j-k}$ for all k; hence, $\sigma^{1+k}(S) = s_{1+k}s_{k+2} \dots s_k = s_{j-k}s_{j-k-1} \dots s_{j-k+1} = \sigma^{j-k}(\bar{S})$. In other words r_{k+1} and r_{j-k} cannot be distinguished. This implies that any deterministic algorithm has a synchronous execution in which r_{k+1} and r_{j-k} start with the same initial value, at every step perform the same operations and receive the same value; thus they remain indistinguishable. Therefore no unique leader can be elected. ■

By lemma 6 and lemma 7, it follows

Theorem 8 *In unoriented rings $VE(S) \equiv MSP(S)$ iff n or $d_0(S)$ is odd.*

By definition of GEP, from theorem 7 and theorem 8, it follows that:

Theorem 9 $MSP \equiv GEP$.

Let us now show how certain values of n can help finding a solution to the GEP and MSP.

Theorem 10 *MSP and VE are deterministically solvable if n is prime.*

PROOF By property 3, theorem 1, theorem 6, theorem 8, and theorem 9. ■

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

Theorem 11 $MSP \geq XOR$.

PROOF First observe that the XOR function is invariant in 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_1 (and $s_{d_0(S)}$ if the ring is undirected) compute $d_0(S)$ (and $d_0(\bar{S})$ if unoriented); then $XOR(S) = XOR(\bar{S}) = [d_0(S)]_{\bmod 2} = [d_0(\bar{S})]_{\bmod 2}$ is trivially computable. ■

4 Sorting and electing a leader in anonymous asynchronous rings

In this section, we show how to use the results related to the properties of the string S and to the value n, in order to solve the GEP and MSP in anonymous asynchronous rings. We consider all boolean strings (i.e., $\delta(S) \leq 2$).

4.1 Oriented rings

In this section, we first present an algorithm that solves the GEP and MSP for the case n prime. As shown later the algorithm is optimal. For the case n not prime, we give another algorithm that will be shown later to lead to a good average case on every input and every value of n.

All the algorithms presented consider $R(S)$ and solve $GEP(S)$ or $MSP(S)$, if a solution exists; in the case no solution exists, all vertices become aware of this fact.

4.1.1 Case n prime

Let us first present an algorithm that is used for the case n prime. It is divided into three steps. In the first one every processor starts in an *active-step 1* state and has an input bit b; different cases may arise: a) it receives a total of $n-1$ bits equal to b and in this case 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) it sends and receives bits until it receives a bit $\neq b$; it chooses as a value the number of b's it has collected plus its own, sends its value to the left and to the right and then moves to an *active-step 2.1* state; c) it receives messages from other processors that are in the next state (active-step 2.1) and in this case it becomes *passive*.

Step 2 is divided into two sub steps: active-step 2.1 and active-step 2.2. In active-step 2.1 state, a processor receives the values of its active neighbours. If its value is a local maxima, it remains active, otherwise it becomes passive. Every active processor p sends a counter to the right and moves to an *active-step 2.2* state. Passive processors that receive this counter increase it and forward it. In active step 2.2 state p eventually receives a counter from the left; if this value is n, p knows that this is its own message and that all the other processors are passive, it becomes *leader* and moves to an *active-step 3* state, otherwise it chooses the value of the counter as its new value and moves back to active-step 2.1. Every passive processor forwards messages and, when appropriate, it increases counters.

Finally, at active-step 3 different actions are taken depending on whether the problem to be solved is the election or sorting. In case of election, the leader sends its value around which transforms all other processors from passive to *defeated* and enters a final state *elected*. In case of sorting, it determines (by circulating a counter) $d_0(S)$, chooses value 0 and it tells the first $d_0(S)-1$ processors on its right to change their bit into 0 and the others into 1; once a processor knows its final value it becomes *sorted*. A formal description of the code of this algorithm is contained in the Appendix.

We now introduce an important definition. 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 and t_j, respectively, let $l(S)$ denote the number of such pairs. Let $z_{max} = \max\{z_j\}$, $t_{max} = \max\{t_j\}$, for $j = 1, 2, \ldots, l(S)$. When no ambiguity arises we will omit the subscripts and use l instead of $l(S)$.

Theorem 12 *The above algorithm correctly solves the GEP and MSP for the case n prime (and MSP also for $S \in \{0^n, 1^n\}$). The algorithm requires the exchange of at most $O(\sum_{j=1}^{l}({z_j}^2 + {t_j}^2) + n \log n)$ messages.*

Proof Let us first prove the correctness. Active-step 1 is trivial: either every processor collects n bits and knows that $S \in \{0^n, 1^n\}$ and therefore that the string is sorted and that a leader cannot be elected, or at least one processor has seen a bit different to the one it has (b) and it assigns itself a value corresponding to the distance to the next value $\neq b$ to its left. Since in this case $S \notin \{0^n, 1^n\}$ at least one processor will move to an active-step 2.1 state. Every processor in active-step 1 state that receives a message from active-step 2.1 state becomes passive. Processors in active-step 2.1 state that are local maxima remain active, send a counter around and move to the active-step 2.2 state. A processor in active-step 2.2 state checks whether the counter has traveled all along the ring (i.e., its value is n); in this case, all other processors must be passive, hence, it becomes leader and chooses the distance to the previous active processor (the value of the counter) as a new value and moves back to the active-step 2.1 state. Note that, since n is prime, then S is not periodic (see property 3), therefore in active-step 2.1 state, the initial values are not all identical, further at least one processor p chooses a new value different from the one of at least one of its neighbours (otherwise the string would trivially be $S = b^n$).

We want to prove now that in active-step 2.1 state at least one processor remains active. Consider the active processor p which has chosen a different value and its two active left and right neighbours p' and p'' with value v' and v'', respectively. If p has a value v bigger than v' and v'', then it remains active. Since the value by assumption cannot be equal to both, if it is smaller than one of them, then p becomes passive. In this case, though, either v' or v'' (or both) are bigger than v. W.l.o.g. consider the case where $v' > v$. If v' is also bigger than the value of its other neighbour it survives, otherwise it becomes passive. Increasing values can be found moving in a counter-clock direction, but since the ring is cyclic and since $v' > v$ there exists at least a processor p''' that contains a value bigger than both of its neighbours therefore it remains active. At every active-step 2.1 and 2.2 at least 1/3 of the processors become passive since a processor remains active if it has a value than the one of at least one of its neighbours (that in this case becomes passive). This implies that these states are repeated at most $\log_{1.5} n = O(\log n)$ times.

At active-step 3 a unique processor is active and it can trivially elect itself a leader, count $d_0(S)$ and sort the string.

For the complexity cost, observe that at active-step 1 every processor collects at most a set z_i or t_i of bits, depending on the block it receives. More precisely every bit 0 (1) in block z_j (t_j) at distance $i < z_j$ from the first bit 1 (0) travels at most i steps, therefore in block z_j at most $\sum_{i=1}^{z_j} i = \frac{z_i(z_i+1)}{2}$ ($\sum_{i=1}^{t_j} i = \frac{t_i(t_i+1)}{2}$) bits travel. Totally at most $\sum_{j=1}^{l}(\frac{z_i(z_i+1)}{2}) + \frac{t_i(t_i+1)}{2})$ 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 active-step 2.1 and 2.2 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 its value. The total number of messages exchanged in this state is then $O(n \log n)$. Finally at active-step 3 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 $O(\sum_{j=1}^{l}(z_i^2 + t_i^2) + n \log n)$. ■

In a later section we will show that this algorithm is optimal.

4.1.2 Case n not prime

Let us now consider the case when n is not prime. If we apply a trivial input collection algorithm we can solve the problem in $O(n^2)$ messages. In fact, if the string is periodic, every processor will detect it; if it is not periodic, a unique maximal lexicographical shift of the string exists (see property 2) and the processor that collects it will be elected as a leader and will do the sorting, if needed.

We now present a good average case (with respect to the input) algorithm. We will use the notation of Kolmogorov random [9]; briefly, an n bit string is Kolmogorov random (k-random for short) if the length of the shortest program outputting the string is longer than $n - \log n$ bits. In this context it is trivial to see that the fraction of Boolean strings of length n that are not k-random is only $1/n$ ($2^{n-\log n}/2^n = 1/n$). For further details refer to [9].

The algorithm that we now present is based on a fact that is derived from some results in [8, 14]:

Property 4 *[8, 14] There exists a constant C such that, all substrings of length $C\log n$ of any n-bit cyclic k-random string are different.*

From this fact it trivially follows:

Property 5 *There exists a constant C' such that, all substrings of length $C'\log n$ of any n-bit cyclic k-random string and of its reverse string are different.*

The main idea of the algorithm is that every processor starts in an *active-step 1* state where it collects at least $C\log n$, but no more than $n-1$ values, continuing as long as receiving values equal to b. If the number of collected values is $n-1$, it moves to an *all-equal* state and it ends, since it detects that $S \in \{0^n, 1^n\}$ and therefore it knows that no leader can be elected and that the string is already sorted. If exactly $C\log n$ values have been collected it uses the received string as a value and moves to an *active-step 2* state; otherwise, it chooses as a value the number of received bits and moves to active-step 2 state. If a processor in active-step 1 state receives an active-step 2 message it becomes passive. All active processors that move to active-step 2 state run a Hirschberg and Sinclair leader election algorithm [7] using the chosen values as identifiers. Since those values might not be distinguished, more than one processor can become leader. The Hirschberg and Sinclair algorithm can be easily modified to handle this case (e.g., by neither stopping, nor "killing" encountered processors with the same value, and continuing for exactly $\lceil \log n \rceil$ steps). Every leader then moves to an *active-step 3* state and sends a counter around to determine if it is unique; in this case it becomes *elected* and eventually sorts. Otherwise, every leader sends a stop bit around and all processors (even the passive one) move to an *active-step 4* state. In this state, every processor runs an input collection. If S is aperiodic, the processor that has collected the unique maximal lexicographical rotation of the string is elected and eventually sorts; if S is periodic, all processor move to an all-equal state and end.

Theorem 13 *The above algorithm correctly solves the GEP and MSP for the case S aperiodic (and MSP also for $S \in \{0^n, 1^n\}$); it requires on average, with respect to all possible input strings, $O(n \log n)$ messages.*

Proof In active-step 1 state, if $S \in \{0^n, 1^n\}$, then every processor detects it and stops. In this case, $O(n^2)$ messages travel around. If the string is k-random we know from property 4 that all substrings of length $C \log n$ are distinct and moreover the string cannot be all 0's or all 1's. Therefore active-step 2 state is reached by all processors after collecting $O(\log n)$ bits and moreover a unique leader will be elected after at most $O(n \log n)$ messages have been exchanged [7]. For all other strings, all processors perform a (partial) input collection; this in the worst case requires $O(n^2)$ messages. Each processor chooses this input as value and moves to active-step 2 state. The values might not be unique, therefore, when Hirschberg and Sinclair algorithm terminates many leaders will be elected; in the worst case $O(n^2)$ messages will have been exchanged. These leaders will though know that they are not unique because, while active-step 3 state, they will receive a counter with value $< n$. If no unique leader is elected then in active-step 4 state all processors do input collection, therefore solving the problem if S is aperiodic (or MSP for $S \in \{0^n, 1^n\}$) since there exists a unique maximal substring, and the processor that collects it is a leader. This requires other $O(n^2)$ messages. The leader computes $d_0(S)$ and sorts as in of algorithm of section 4.1.1 and at most other $O(n)$ messages travel around. Summarizing, the number of messages exchanged is $O(n \log n)$ for the case of k-random strings, $O(n^2)$ in the worst case otherwise. Since the fraction of strings that are not k-random is at most $1/n$ we have that on average the number of messages exchanged is:

$$1/n \, O(n^2) + (1 - 1/n) \, O(n \log n) = O(n \log n).$$

■

4.2 Unoriented rings

In this section we show how to modify the given algorithms for oriented asynchronous rings to solve the GEP and MSP for unoriented rings.

4.2.1 Case n prime

Let us first consider the case of n prime. The main idea is that every processor can run separate executions of algorithm of section 4.1.1 in both directions, and as a result, two leaders are elected if $S \notin \{0^n, 1^n\}$, or it is detected that $S \in \{0^n, 1^n\}$. If $\delta(S) = 2$, the two leaders (by each sending counters) compute the two distances c and c' (n is known). Since one of them is even and the other odd, they send an election message to the processor in the middle of the path of the even distance; this processor moves to an *elected* state and can eventually sort. As usual, passive processor forward (and, if appropriate, increase) the received values.

Observe that, since the algorithm of section 4.1.1 is executed concurrently a processor can be in different states with respect to each execution. Notice that it is possible that a passive processor becomes elected (because it is in the middle of the path).

Theorem 14 *The above algorithm correctly solves the GEP and MSP for the case n prime and S aperiodic (and MSP also for $S \in \{0^n, 1^n\}$) in an anonymous unoriented ring and it requires $O(\sum_{j=1}^{l}(z_j{}^2 + t_j{}^2) + n \log n)$ messages.*

PROOF The correctness trivially derives from the one of theorem 12 and from the fact that, since n is prime, only one of c and c' is odd. The message complexity is twice the one of algorithm of section 4.1.1; the extra step requires at most $O(n)$ messages. ■

4.2.2 Case n not prime

Let us now consider the case n not prime.

We now give a good average case (with respect to the input) algorithm. The main idea is that every processor runs algorithm of section 4.1.2 simultaneously and separately in both directions. If $S \in \{0^n, 1^n\}$ or S is periodic, the algorithm will stop detecting such cases, otherwise it elects two leaders v and y which move to a new state. In this state if $v > y$ (or $y > v$) then v (y) is elected, otherwise the right and left distances c and c', between v and y are computed; if $c \neq c'$ and either c or c' is odd the processor in the middle of the smallest odd distance is elected. If both c and c' are even or equal run an input collection algorithm. Once the strings have been collected, the two leaders can determine whether there exists a vertex which is unambiguously determinable in S and $\bar{S}$; if so, such a vertex becomes the only elected vertex. Otherwise, VE is not solvable but an edge can be uniquely determined in S and $\bar{S}$ since the string is aperiodic.

Theorem 15 *The above algorithm correctly solves the GEP and MSP for S aperiodic (and MSP also for $S \in \{0^n, 1^n\}$), and requires on the average, with respect to all possible input strings, the exchange of at most $O(n \log n)$ messages.*

PROOF The correctness derives from the one of algorithm of section 4.1.2 and from the fact that the computations in both directions are separated. If the string S is periodic and the algorithm detects it stops. If the string S is aperiodic (therefore also $\bar{S}$) a unique leader is elected in both directions.

If a vertex can be elected immediately (by the checks on n, c, and c') the algorithm does so. Otherwise, both leaders do complete input collection which enables them to unambiguously determine the solution to GEP. The complexity depends on the one of algorithm of section 4.1.2. If the string k-random, this last algorithm requires only $O(n \log n)$ messages; in this the values of the two leaders are distinct (by property 5), hence one of them will be elected with an $O(n)$ cost. If the string is not k-random, algorithm of section 4.1.2 requires on in the worst case $O(n^2)$ messages, and so does the input collection performed by the two leaders. ■

5 Lower bounds

In this section we study the lower bounds for the asynchronous (un)oriented rings.

We first have to define the following family:

Definition 1 *Given a string S consisting of alternating strings of 0's and 1's, z_i and t_i, we call $F(S)$ the family of all the boolean functions $f : S \rightarrow \{0,1\}$, $f \in F(S)$, such that $\forall i \leq z_{max}$, $\forall j \leq t_{max}$, $\exists c_1, c_2$ $\{0,1\}^{n-i}$, $\exists c'_1, c'_2 \in \{0,1\}^{n-j}$, such that: $f(0^i c_1) = f(0^n)$, and $f(0^i c_2) \neq f(0^n)$; $f(1^j c'_1) = f(1^n)$, and $f(1^j c'_2) \neq f(1^n)$.*

Lemma 8 *Given a string S, the computation of every boolean function $f \in F(S)$ in an anonymous (un)oriented ring, requires at least $\Omega(\sum_{j=1}^{l}({z_j}^2 + {t_j}^2))$ messages.*

Proof Consider the process in a single block of length z_i, and the block of $\frac{z_i}{3}$ processors in the middle for the oriented case ($\frac{z_i}{2}$ on the right for the unoriented case). Assume also that there exists a bad adversary that synchronizes the steps but delays messages that try to move outside the block z_i. Since the view of all the $\frac{z_i}{3}$ processors i (of $\frac{z_i}{2}$ on the right respectively) is the same for at least $\frac{z_i}{3}$ ($\frac{z_i}{2}$ respectively) steps, then all processors will execute the same move during this time. If a processor i computes the function $f \in F$ before $\frac{z_i}{3}$ ($\frac{z_i}{2}$) steps, then the adversary can choose how to complete the string with the bits not jet seen by i so that the output of the function is different. This implies that $\frac{z_i}{3}$ ($\frac{z_i}{2}$) steps must be performed.

After all these steps, processors in the block might receive a message from the processors in the border (the one on the left for the oriented case). At this point, the adversary can remove the delay but at least $(\frac{z_i}{3})^2$ $((\frac{z_i}{2})^2)$ messages will have been exchanged.

This holds for every block z_i and t_j therefore at least $\Omega(\sum_{j=1}^{l}({z_j}^2 + {t_j}^2))$ messages have to travel.

Theorem 16 *Given a string S the GEP and MSP on S require at least $\Omega(\sum_{j=1}^{l}({z_j}^2 + {t_j}^2) + n \log n)$ messages in an asynchronous ring.*

Proof We split the bound in two parts: the $\Omega(n \log n)$ bound is derived from the classical lower bounds for the leader election problem in asynchronous rings. We can prove that the XOR function belongs to the family F and therefore its computation requires $\Omega(\sum j =_1{}^l(z_j{}^2+t_j{}^2))$ messages (see lemma 8) We now prove that $XOR \in F$. Let c_1 and c_2 (c'_1 and c'_2) be two strings of length $n-i$ ($n-j$ respectively) which differ exactly in one $XOR(0^i c_1) \neq XOR(0^i c_2)$ and $XOR(1^j c'_1) \neq XOR(1^j c'_2)$. Then $x \in F$.

From theorem 11 we know that the lower bound for the XOR function is a lower bound for the GEP and from theorem 9 also for the MSP hence the claimed bound. ■

Corollary 2 *By theorem 12, theorem 14, theorem 16, algorithm of section 4.1.1 and algorithm of section 4.2.1 are optimal.*

We finally prove the following:

Lemma 9 *For almost all input strings S, $\sum_{j=1}^{l}(z_j{}^2+t_j{}^2) \in O(n \log n)$.*

PROOF It trivially follows from the fact that almost all strings are k-random and for these strings $\sum_{j=1}^{l}(z_j{}^2+t_j{}^2) \leq cn \log n$, c constant. PROOF

Corollary 3 *For almost all input strings the lower bound of theorem 16 reduces to $\Omega(n \log n)$ messages.*

PROOF Follows directly from lemma 9. ■

References

[1] P. Alimonti, P. Flocchini, and N. Santoro. Finding the extrema of a distributed multiset of values. *Journal of Parallel and Distributed Computing*, 37:123–133, 1996.

[2] 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.

[3] H. Attiya and Y. Mansour. Language complexity on the synchronous anonymous ring. *Theoretical Computer Science*, 53:169–185, 1987.

[4] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 4(35):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*, 1(108):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 Colloquium on Structural Information and Communication Complexity*, pages 35–48, 1994.

[7] D.S. Hirschberg and J.B. Sinclair. Decentralized extrema-finding in circular configurations of processors. *ACM TOPLAS*, 5:627–628, 1980.

[8] E. Kranakis, D. Krizanc, and F.L. Luccio. String recognition on anonymous rings. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science*, pages 392–401, 1995.

[9] M. Li and P.M.B. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, 1997.

[10] M.C. Loui. The complexity of sorting on distributed systems. *Information and Control*, 60(1-3):70–85, 1984.

[11] N. Lynch. *Distributed Algorithms*. Morgan-Kaufmann, 1995.

[12] S. Moran and M.K. Warmuth. Gap theorems for distributed computation. In *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*, pages 131–140, 1986.

[13] D. Rotem, N. Santoro, and J.B. Sidney. Distributed sorting. *IEEE Transactions on Computers*, 4(34):372–376, 1985.

[14] J. Seiferas. A simplified lower bound for context-free language recognition. *Information and Control*, 69:255–260, 1986.

[15] M. Yamashita and T. Kameda. Computing on anonymous networks, part I: Characterizing the solvable cases. *IEEE Transaction on Parallel and Distributed Computing*, 1(7):69–89, 1996.

Appendix

Algorithm for the case: n prime

```
INITIAL
        become active;
        send b to right;
        c := 1;

ACTIVE_STEP1
        receive ⟨S1, x⟩
                if x ≠ b then
                                v := c;
                                send ⟨S2.1, v⟩ to both directions;
                                become ACTIVE_STEP2_1
                        else
                                c := c + 1;
                                if  c = n then
                                                become ALLEQUAL
                                        else
                                                send ⟨S1, b⟩ to right
                                fi
                fi;
        receive ⟨S2.1, value⟩ from direction;
                send ⟨S2.1, value⟩ to opposite direction;
                become PASSIVE;

ALLEQUAL
        stop

ACTIVE_STEP2_1
        receive x from  one direction;
        receive y from  the opposite direction;
        if x = y = v or x > y or y > v then
                                                become PASSIVE
                                        else
                                                send ⟨S2.2, d := 0⟩ to right;
                                                become ACTIVE_STEP2_2;
        fi

ACTIVE_STEP2_2
        receive  d from left;
        if d = n then
                        become ACTIVE_STEP3 (leader);
                else
```

```
                    v := d;
                    send ⟨S2.1, v⟩ to both directions;
                    become ACTIVE_STEP2_1;
      fi

ACTIVE_STEP3
if Problem.type=Election then
                        send ⟨E, value⟩ to right;
                        become ELECTED
                    else /* Problem.type=Sorting */
                        send ⟨SOR.1, counter := 0⟩ to right;
                        become SORTING

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

PASSIVE
receive ⟨HEAD, value⟩ from direction;
if HEAD = S2.1 then
                    send ⟨HEAD, value⟩ to opposite direction;
if HEAD = S2.2 then
                    value := value + 1;
                    send ⟨HEAD, value⟩ to opposite direction;
if HEAD = E then
                  send ⟨HEAD, value⟩ to opposite direction;
                  become DEFEATED
if HEAD = SOR.1 then
                    if b = 0 then value := value + 1;
                    send ⟨HEAD, value⟩ to opposite direction;
if HEAD = SOR.2 then
                    if value > 0 then b := 0
                              else b := 1;
                    send ⟨HEAD, value⟩ to right;
                    become SORTED

SORTED
stop
ELECTED
stop
DEFEATED
stop
```