Finding the Extrema of a Distributed Multiset *

Paola Alimonti †

Dip. Informatica Sistemistica

Università di Roma "la Sapienza", Italy.

alimonti@dis.uniroma1.it

Paola Flocchini Nicola Santoro

School of Computer Science

Carleton University, Canada.

{flocchin,santoro }@scs.carleton.ca

Abstract

We consider the problem of finding the extrema of a distributed multiset in a ring; that is, of determining the minimum and the maximum values, x_{min} and x_{max}, of a multiset $X = \{x_0, x_2, ..., x_{n-1}\}$, whose elements are drawn from a totally ordered universe U and stored at the n entities of a ring network.

This problem is unsolvable if the ring size is not known to the entities, and has complexity $\Theta(n^2)$ in the case of asynchronous rings of known size.

We show that, in synchronous rings of known size, this problem can always be solved in $O((c + \log n) \cdot n)$ bits and $O(n \cdot c \cdot x^{\frac{1}{c}})$ time for any integer $c > 0$, where $x = Max\{|x_{min}|, |x_{max}|\}$. The previous solutions required $O(n^2)$ bits and the same amount of time.

Based on these results, we also present a bit optimal solution to the problem of finding the multiplicity of the extrema.

Keywords: Extrema Finding, Distributed Multiset, Bits-Time Tradeoff, Anonymous Rings, Synchronous Protocols.

*Work partially supported by: ESPRIT III, Basic Research Action Program ALCOM 2; Ministero dell'Università e della Ricerca Scientifica e Tecnologica (Italy), Project "Algoritmi, Modelli di Calcolo e Strutture Informative"; Consiglio Nazionale delle Ricerche (Italy); and Natural Sciences and Engineering Research Council (Canada), under grant A2415.

†This research was done while this author was visiting Carleton University

1 Introduction

Let $X = \{x_0, x_2, ..., x_{n-1}\}$ be a multiset, whose values are drawn from a totally ordered universe U and stored at the processors (or entities) of a ring network. Let i denote the ring entity where element x_i is stored.

The *Extrema-Finding Problem* (EF) is the problem of finding the minimum and the maximum values, x_{min} and x_{max}, in X; that is, of each entity i determining whether $x_i = x_{min}$, $x_i = x_{max}$ or $x_{min} < x_i < x_{max}$.

The simplicity of the statement of the problem is deceiving. In fact, its complexity depends on many factors of totally different nature.

If X is a *set* (i.e., all values are distinct), EF is computationally equivalent to the *Leader Election Problem* (LE): any extrema-finding algorithm directly solves LE (e.g., the unique minimum becomes leader); conversely, if there is a leader, x_{min} and x_{max} can be found with a lower communication complexity than that required for leader election. If the ring is *asynchronous*, the number of messages required to solve LE (and, thus, EF) is $\Theta(n \cdot logn)$ [4, 6, 7, 8, 14, 15]. For *synchronous* rings, several solutions have been presented with different bits-time complexities [5, 8, 9, 10, 11, 16, 18, 19]. Unlike the asynchronous case, knowledge of the ring size n has an impact on the established upper bounds. When n is known, the best bits-time bound for LE (and, thus, EF) obtained so far is $O(c \cdot n)$ bits and $O(c \cdot n \cdot v^{\frac{1}{c}})$ time [10], where c is an arbitrary positive integer and $v = Min\{|x_{min}|, |x_{max}|\}$. When n is not known, the best bits-time tradeoff is $O(n \cdot g^{-1}(n))$ bits and $O(v \cdot g(g^{-1}(n))$ time, where g is an arbitrary monotonically super-increasing function and g^{-1} is the pseudo-inverse of g [13]. In all these results it is assumed that $U = \mathbf{Z}$ (the set of positive integer).

The situation is more complex if the values are not necessarily distinct. In this case, which is also called the *anonymous ring* case, EF is *unsolvable* (i.e., no deterministic solution protocol exists) if the ring size n is not known to the entities [2]. Thus, let us assume that n is known.

For *asynchronous* rings $\Omega(n^2)$ messages are required [2], and this bound can be trivially achieved.

In *synchronous* rings, if $U = \mathbf{Z}^+$ (i.e., the values are positive integers), it is possible to find the minimum value with $O(n)$ bits; furthermore, this can be done with a time sublinear in x_{min} [10]. No similar results exist for maxima finding. Note that to restrict the values to be positive integers is equivalent to assuming: (i) the existence of a lower bound on the minimum value, and (ii) the knowledge by the entities of such a bound. In this light, the existing results can be rephrased as follows: if $U \subset \mathbf{Z}$ has a minimum u_{min} and the minimum is known to the entities, then x_{min} can be found using $O(c \cdot n)$ bits and $O(c \cdot n \cdot w^{\frac{1}{c}})$ time for any integer constant $c > 0$, where $w = x_{min}$ or $w = x_{min} - u_{min}$ depending on whether or not $(u_{min} \cdot x_{min}) < 0$, respectively. Clearly, if U has a maximum u_{max} which is known to the entities, also maxima finding can be solved with similar bits-time bounds.

It follows that, if U is *finite* and its minimum and maximum are both known to the entities, EF can be solved in $O(n)$ bits and a time which is sub-linear in $|U|$. This represents an improvement over the input collection technique of [2] which would require $O(n \cdot logn)$ bits and time linear in $|U|$. See Table 1 for a summary. Note that n (the ring size) is a system parameter; on the other hand, the values $|U|$, x_{min} and x_{max} are input parameters (i.e. a priori unbounded).

Ref.	Bits	Time	U
[2]	$O(n \log n)$	$O(\log n \cdot u)$	finite, u_{min} & u_{max} known
[10]	$O(c \cdot n)$	$O(c \cdot n \cdot u^{\frac{1}{c}})$	finite, u_{min} & u_{max} known
[12]*	$O(c \cdot n^2)$	$O(c \cdot n \cdot x^{\frac{1}{c}})$	infinite or unknown bounds
[10]*	$O(c \cdot m \cdot n)$	$O(c \cdot m \cdot n \cdot x^{\frac{1}{c}})$	infinite or unknown bounds
this paper	$O((c + \log n) \cdot n)$	$O(c \cdot n \cdot x^{\frac{1}{c}})$	infinite or unknown bounds

Table 1 - Summary of results for EF, in synchronous rings of known size, where $x = Max\{|x_{min}|, |x_{max}|\}$, m is the number of distinct values in X, $c > 0$ is an arbitrary integer constant, $u = |U|$.

(* bounds implied by the results in the reference)

An interesting interpretation of these results is the following: if U is finite and its minimum and maximum are both known to the entities, then EF can be solved with the same complexity as if U was a set. In other words, for synchronous rings of known size, *a priori knowledge of a bound on the range* of the values in the multiset has the same "power" as *distinctness* of the values.

Several questions immediately and naturally arise: what happens if we do not have this a priori knowledge? what happens if these bounds do not exist (e.g., $U = \mathbf{Z}$)? In other words, what is the complexity of EF?

Some upper bounds on the complexity of EF can be derived starting from the existing results. Using the two-parties communication protocol of [12], the synchronous transformation of the (obvious but optimal) $O(n^2)$ messages asynchronous algorithm yields a general solution using $O(c \cdot n^2)$ bits and $O(c \cdot n \cdot x^{\frac{1}{c}})$ time, where $x = Max\{|x_{min}|, |x_{max}|\}$. A slight improvement in the bit complexity can be obtained by repeated applications of the minima finding technique of [10] with simple modifications. This would yield a general solution using $O(c \cdot n \cdot m)$ bits at the expenses of a $O(c \cdot m \cdot n \cdot x^{\frac{1}{c}})$ time complexity, where m is the number of distinct elements in X. In the worst case, this is still a $O(n^2)$ bits solution. (See Table 1 for a summary.) An immediate question is: can EF be solved in $o(n^2)$ bits and, if so, what is the corresponding time complexity ?

The main contribution of this paper is a positive answer to this question. Namely, we prove the following:

Main Theorem *For any a priori choice of an integer $c > 0$, EF can be solved with $O((c + \log n) \cdot n)$ bits in time $O(c \cdot n \cdot x^{\frac{1}{c}})$, where $x = Max\{|x_{min}|, |x_{max}|\}$.*

We prove this theorem by exhibiting an algorithm with the specified complexity. Furthermore, if the values are distinct, the bit complexity of the algorithm is $O(c \cdot n)$, matching the best bounds available for those cases, but *without* a priori knowledge of the distinctness of the values.

At the core of this result is a novel maxima finding algorithm for $U = \mathbf{Z}^+$. This algorithm uses a variety of well known techniques (*Waiting*, *Counting*, *Pipeline*) and requires $O((c + logn) \cdot n)$ bits and $O(c \cdot n \cdot x_{max}^{\frac{1}{c}})$ time for any integer $c > 0$.

We also consider the *Extrema Counting Problem* (EC); that is, the problem of finding the multiplicity of the extrema. We show how the new algorithm for EF immediately implies the same bits-time upper bounds for EC; we also prove that the resulting solution is bitwise optimal.

All these results hold for both unidirectional and bidirectional rings.

The paper is organized as follows. In the next section, the terminology and notations are introduced and the overall structure of the solution to EF, for unidirectional rings, is presented. In Section 3, a novel maxima-finding algorithm for $U = \mathbf{Z}^+$ is presented. In Section 4, its correctness is proved, and its complexity is analyzed. The time complexity is further reduced in Section 5 using a technique which leads to the desired tradeoff and the proof of the Main Theorem. In Section 6, it is described how to apply the proposed extrema finding algorithm to solve EC; a lower bound on the bit complexity of EC is shown, proving that the obtained solution is bitwise optimal. Finally, in Section 7, we show how to obtain the same bounds when the ring has bidirectional links, and conclude by posing a conjecture.

2 Extrema Finding

2.1 Definitions and Basic Strategy

The model we consider is a *synchronous ring* of n entities; the size n of the ring is known to the entities. In the following we will consider *unidirectional* links; that is, each entity i can send messages to $i+1$ and can receive messages from $i-1$, where all the operations are modulo n. Let us stress that the indices $i-1$, i, $i+1$ are used here for notation purposes, and are not known to the entities. Each entity i has access to a special variable c_i, called *local clock*, which can be reset. All local clocks are synchronized; that is, there exists a global system clock (not accessible to the entities) and the values of all local clocks are incremented by one at every tick of the global clock. Note that the local clocks do not necessarily have the same value. Transmission and queueing delays are bounded by the interval between two successive ticks; that is, every message sent at a global time instant t will be received and processed at the global time t+1; at any time instant, each entity can transmit only one message to its neighbor.

Every entity i has associated a value $x_i \in U$, where U is a totally ordered countable universe; in the following we will consider $U = \mathbf{Z}$. The values are not necessarily distinct; that is, $X = \{x_0, x_1, ..., x_{n-1}\}$ is a multiset. Let x_{min} and x_{max} denote the minimum and maximum values in X, respectively. The *Extrema Finding Problem* (EF) is the problem of each entity i determining whether $x_i = x_{min}$, $x_i = x_{max}$ or $x_{min} < x_i < x_{max}$.

Let $Max\langle X; U'\rangle$ and $Min\langle X; U'\rangle$ denote the problem of determining x_{max} and x_{min}, respectively, when the elements of X are drawn from $U' \subseteq \mathbf{Z}$. To obtain a general solution to EF is obviously sufficient to provide a solution to both $Max\langle X; \mathbf{Z}\rangle$ and $Min\langle X; \mathbf{Z}\rangle$. As mentioned in the introduction, an efficient solution exists for $Min\langle X; \mathbf{Z}^+\rangle$ [10].

The overall strategy we use to solve EF is the following:

Basic Strategy

1. Determine whether: (i) all values in X are positive; (ii) all values in X are negative; (iii) in X there are both positive and negative values. This can be easily accomplished by computing both the *AND* and the *OR* on the Boolean multiset $B = \{b_0, b_1, ..., b_{n-1}\}$, where $b_i = 1$ iff $x_i \geq 0$. If the result of the *AND* is 1, then we are in case (i); if the result of the *OR* is 0, then we are in case (ii); if the result of the *AND* is 0 and the result of the *OR* is 1, then we are in case (iii).

2. In case (i), solve $Min\langle X; \mathbf{Z}^+\rangle = Min\langle X; \mathbf{Z}\rangle$ and $Max\langle X; \mathbf{Z}^+\rangle = Max\langle X; \mathbf{Z}\rangle$. In case (ii), let $X' = \{|x_0|, |x_1|, ..., |x_{n-1}|\}$; solve $Min\langle X'; \mathbf{Z}^+\rangle = Max\langle X; \mathbf{Z}\rangle$ and $Max\langle X'; \mathbf{Z}^+\rangle = Min\langle X, \mathbf{Z}\rangle$. As for case (iii), let $P = \{p_0, p_1, ..., p_{n-1}\}$ and $Q = \{q_0, q_1, ..., q_{n-1}\}$ be the multisets on $\mathbf{Z} \cup \{*\}$ defined as follows: if $x_i < 0$ then $p_i = *$ and $q_i = |x_i|$, otherwise $p_i = x_i$ and $q_i = *$. Then $Max\langle P; \mathbf{Z}^+ \cup \{*\}\rangle = Max\langle X; \mathbf{Z}\rangle$ and $Max\langle Q; \mathbf{Z}^+ \cup \{*\}\rangle = Min\langle X; \mathbf{Z}\rangle$, where any entity with value $*$ acts only as a relayer.

In other words, once the *AND* and *OR* have been computed, EF can be reduced to the computation either of a minima and a maxima finding, or of two maxima finding, and always in a multiset with values in $\mathbf{Z}^+$. In $\mathbf{Z}^+$, the bits-time complexity of finding the maxima is not smaller than that of finding the minima (since, clearly, knowledge of a bound can not worsen the complexity). The computation of the *AND* and *OR* can be done in $O(n)$ bits and time (e.g.,[2]); thus it does not affect the final complexity. Therefore,

Theorem 2.1 *Let A be an algorithm which, for any multiset Y with values in $\mathbf{Z}^+$, solves* $\mathrm{Max}\langle Y; \mathbf{Z}^+\rangle$ *with* $\mathrm{B_A}(Y)$ *bits and* $\mathrm{T_A}(Y)$ *time. Then there is an algorithm which, for any* $X = \{x_0, ..., x_{n-1}\}$ *with values in* $\mathbf{Z}$*, solves* $EF\langle X; \mathbf{Z}\rangle$ *with* $O(\mathrm{B_A}(X'))$ *bits and* $O(\mathrm{T_A}(X')$ *time, where* $X' = \{|x_0|, ..., |x_{n-1}|\}$*.*

2.2 Basic Techniques

As discussed before, to efficiently solve EF is sufficient to efficiently solve $Max\langle X; \mathbf{Z}^+\rangle$. We propose a solution which uses three well-known techniques (Waiting, Counting and Pipeline) briefly described in the following.

Waiting is a basic strategy to determine whether a desired condition is verified. We use the following Waiting protocol: each entity i waits $f(a_i, b_i)$ units of time, where a_i and b_i are values known to i. If, while waiting, a message is received, i forwards it and becomes "large"; otherwise, it sends a message and becomes "minimum". The quantity $f(a_i, b_i)$ is called *timeout value*; an appropriate choice of f, a_i and b_i guarantees that the protocol correctly finds the minimum value among the executing entities, provided that they start the execution with bounded delay. Variations of this protocol have been used for minima finding and leader election in synchronous rings [9, 11, 13, 16].

The *Counting* technique (also called a *communicator*) is any protocol which solves the *Two-Parties Communication Problem* (TPC). TPC is the problem of an entity (the *sender*) communicating information from an unbounded, but countable, universe U to a neighboring entity (the *receiver*) using a combination of time and *silence* (i.e., absence of transmission). It is usually assumed that $U = \mathbf{Z}^+$. The communicator we use is the Two-Bits communicator: the sender communicates x by transmitting a bit, waiting x units of time, and transmitting another bit; the receiver reconstructs x from the difference of the reception times of the two bits. One of the first uses of a communicator (for finite U) in synchronous rings can be found in [1]; the Two-Bits protocol has been employed in [3, 5, 13], and a more complex communicator has been described in [17]. Communicators achieving optimal bits-time complexity have been recently derived [12].

Pipeline is the classical technique extensively used in systolic and parallel computing: in a chain of neighboring entities (called *segment*) and starting from the entity at the beginning of the segment (the *head*), the Pipeline protocol accumulates at the entity at the end of the segment (the *accumulator*) some function of the values held by the entities. This technique has been used in synchronous rings by [2, 3, 17].

3 A Maxima Finding Protocol

In this section we present a protocol for $Max\langle X, \mathbf{Z}^+\rangle$ which uses $O(n \cdot \log m_{min})$ bits and $O(n \cdot x_M)$ time, where $x_M = x_{max}$ and m_{min} is the multiplicity of x_{min} in X.

It is assumed that, prior to the execution of the protocol, all entities are *sleeping*. A *sleeping* entity starts the execution of the protocol either spontaneously (i.e., receiving a locally originated request) or upon reception of a "wake-up" message from a neighbouring entity. In either cases, it sends a "wake-up" message along the ring, and starts the protocol by becoming *waiting*. It is easy to see that within at most $n-1$ time units from the time the first entity becomes *waiting*, all the entities have started the execution of the protocol. Notice that there is no a priori restriction on the number of entities which spontaneously start the protocol.

The protocol comprises of two phases: Iteration and Notification.

The Iteration phase is a sequence of steps. Informally, at each step of the algorithm, the *waiting* entities holding the minimum values become *players*. This is done by means of the Waiting protocol. The ring is thus divided into disjoint segments, each one delimited by two *players*. The *player* located at the beginning of a segment starts the accumulation at the other *player* of the maximum value (local maximum) in that segment; it will also accumulate the local maximum of the previous segment. The accumulation is done using Pipeline and the communication of values is done using the Two-Bits protocol. Once the accumulation in a segment is completed, all entities in that segment are *passive*; the entity delimiting it holds the local maximum, and starts the next iteration, becoming *waiting* with this value. The Iteration phase terminates when all the *waiting* entities have the same value; in this case, as proved later, this value is x_M. At this point, the *waiting* entities become *done* and all the other entities are *passive*. *Done* entities start the Notification phase.

In the Notification phase, the entities in a segment delimited by two *done* are notified of x_M through a Pipeline started by the *done* entity located at the beginning of that segment. Upon communication of x_M, an entity i becomes either *maximum* (if $x_i = x_M$) or *small*.

The algorithm contains mechanisms to: detect termination of the Iteration phase; ensure that the number of *players* is at least halved at each step; ensure that at each step the difference in initiation time between any two entities which became *players* is bounded; ensure that x_M is always held by a non-*passive* entity.

During the execution of the protocol, an entity can be in one of the following states: {*waiting*, *player*, *pipeline*, *head*, *accumulator*, *delaying*, *suspended*, *done*, *passive*, *maximum*, or *small*}.

Following is the formal description of the algorithm at entity i; the variables used are:
- *clock* is the local clock, initially set to 0.
- v contains the value associated to the entity.
- v_c contains the value of the local maximum of the current step. Initially, it is set to x_i.
- v_p contains the value of the local maximum of the previous step. Initially, it is set to 0.
- t, val are auxiliary variables.

The *when* statement has precedence over the *receiving* statement, should the events occur simultaneously.

(1) WAITING

```
when $clock = f(n, v_c - v_p)$ do
  send <starting:= 0>
  reset $clock$
  become($player$)
receiving <starting> do
  reset $clock$
  if <starting>= 1
     send <starting>
     become($pipeline$)
  if <starting>= 0
     send <starting:= 1>
     become($head$)
```

(2) PLAYER

```
receiving <starting> do
  if <starting>= 0
     reset $clock$
     become($suspended$)
  if <starting>= 1
     $t := clock$
     send <alarm>
     become($accumulator$)
```

(3) ACCUMULATOR

```
receiving <ending> do
   $v_p = v_c$
   $v_c = t - clock$
   become($delaying$)
receiving <starting> do
   send <starting>
   become($passive$)
```

(4) DELAYING

```
when $clock = v_c + 2 \cdot (n - 1)$ do
  become($waiting$)
receiving <starting> do
  reset $clock$
  if <starting>= 1
     send <starting>
     become($pipeline$)
  if <starting>= 0
     send <starting:= 1>
     become($head$)
```

(5) PIPELINE

```
receiving <ending> do
   if $v_c \leq clock$
      send <ending>
      become($passive$)
   else become($head$)
receiving <starting> do
  reset $clock$
  send <starting>
```

(6) HEAD

```
when ($clock = v_c$) do
   send <ending>
   become($passive$)
receiving <starting> do
  reset $clock$
  if <starting>= 1
     send <starting>
     become($pipeline$)
  if <starting>= 0
     send <starting:= 1>
```

(7) SUSPENDED

```
when $clock = n$ do
   send <start_notification>
   reset $clock$
   become($done$)
receiving <alarm> do
   send <alarm>
   become($passive$)
```

(8) PASSIVE

```
receiving($val$) do
   send($val$)
   if $val$ =<start_notification>
      reset $clock$
   if $val$ =<end_notification>
      if $v = v_c$ become($maximum$)
      else become($small$)
```

(9) DONE

```
when $clock = v_c$ do
  send <end_notification>
  if $v = v_c$ become($maximum$)
  else become($small$)
```

We shall now discuss the states in some details. In the following, the states *waiting*, *player*, *pipeline*, *head*, *accumulator*, and *delaying* are called "active".

Waiting entities start new steps executing the Waiting protocol. A *waiting* entity waits $f(n, v)$ time units, where v is a quantity known to the entity (initially set to its value). If, during this time, it does not receive any message, it holds the minimum among the values held by the *waiting* entities. In this case, it sends a <starting> bit, that starts the Pipeline in the segment, and becomes *player*. If, while *waiting*, it receives a <starting> bit, there is a smaller value held by a *waiting* entity; thus, it becomes *head* if it is the first active entity encountered by the <starting> bit, *pipeline* otherwise. This situation is detected by the value of the <starting> bit.

When a **player** receives the <starting> bit sent by the neighbouring *player*, two different state transitions can arise. If there are no active entities in the segment between the two *players*, it becomes *suspended*; otherwise it sends an <alarm> bit to notify that it is still playing, and becomes *accumulator*. Which of the two situations has occurred is detected by the value of the <starting> bit. As a consequence, if a segment consists only of *passive* entities, al least one of the two *players* delimiting it becomes *passive*. This ensures that the number of *players* is at least halved at each step.

Pipeline and **head** entities execute the Pipeline protocol. The purpose of this protocol is to accumulate in the *accumulator* entities the local maxima of the segments. The *head* is the first among the entities performing the Pipeline in a segment. When a *head* entity ends its communication, it sends an <ending> bit and becomes *passive*. When a *pipeline* entity receives the <ending> bit, it compares its held value with the one accumulated so far. If its value is bigger, it becomes *head*; otherwise, it immediately forwards the bit, and becomes *passive*.

An **accumulator** is the entity located at the end of a segment; the value communicated to it by the Pipeline is the maximum value in the segment. When an accumulator receives the <ending> bit, it calculates the value of the local maximum, and become *delaying*.

Since the amount of time required for the Pipeline procedure directly depends on the accumulated values, it can be the case that an entity in state *head*, *pipeline* or *accumulator* at step k receives a <starting> bit from a *player* already in step $k + 1$. In this case, if *head* or *pipeline*, the entity forwards the <starting> bit, and restarts the Pipeline for step $k + 1$ without changing state. If *accumulator*, it forwards the <starting> bit and becomes *passive*; the value it was accumulating is being held either by a *pipeline* or by a *head* entity.

Delaying entities wait a certain amount of time to ensure that, in the next step, any two *waiting* entities, which hold the same value, start the step with a delay bounded by their distance. By design, the only message a delaying entity might receive is a <starting> bit. If, during this time, it does not receive it, it becomes *waiting* . Otherwise, it forwards the <starting> bit and becomes either *head* or *pipeline* depending on whether or not it is the first active entity encountered by the bit. The situation is detected by the value of the bit.

A **suspended** entity waits n time units. If nothing occurs during this time, it becomes *done*. Otherwise, it becomes *passive*.

Done entities know the value of the maximum x_M and start the Notification phase. Each *done* entity i sends a <start_notification> bit, waits x_M time units, and sends an <end_notification> bit. If $x_i = x_M$, it becomes *maximum*, otherwise it becomes *small*.

Passive entities just forward received bits. If the received bit is a <start_notification>, a passive entity i starts counting. When it receives an <end_notification>, it knows x_M. If $x_i = x_M$, it becomes *maximum*, otherwise it becomes *small*.

Maximum and **small** are terminal states for the algorithms.

The state transition diagram is shown in Figure 1.

4 Analysis of the Algorithm

In this section, we show the correctness of the algorithm and analyze its complexity. Since the Notification phase is just a comunication of the values accumulated at the *done* entities, to prove the correctness of the proposed solution, it sufficies to show the correctness of the Iteration phase; that is, the *done* entities hold the maximum value when the Iteration phase terminates.

In the following we will denote by: $d(i,j)$ the distance between entity i and entity j; x^k_{min} the minimum value held by the *waiting* entities at step k; $\Delta^k = x^k_{min} - x^{k-1}_{min}$ where $x^0_{min} = 0$; r^k_i the segment between two successive *players* terminating in *player* i at step k; $x^k_{M_i}$ the maximum value in r^k_i; T^k_i the time when entity i starts step k with respect to the global clock.

We say that a set S of entities start the k-th step with *bounded delay* if, for any $i,j \in S$, $T^k_i \leq T^k_j + d(j,i)$. Let $f(n,x) = 2 \cdot n \cdot x$ be the *timeout function* used by the algorithm.

The proof of correctness is carried out by a sequence of lemmas. The first lemma states a sufficient condition for termination.

Lemma 4.1 *If at step k all* players *become* suspended*, then all the other nodes are* passive*, and the Iteration phase terminates.*

Proof. If a *player* i becomes *suspended*, then in r^k_i there are no *head* entities since any entity which is or becomes *head* sets the <starting> bit to 1 (Rules (1), (4), (6)); similiarly, there are no *pipeline* entities in r^k_i since, by the same rules, to become *pipeline* it is necessary to receive a <starting> bit equal to 1. Moreover, if all *players* become *suspended*, none of them becomes *accumulator* (Rule (2)); hence no entity can become *waiting*. Since only *waiting* entities can start new steps, the Iteration phase terminates at step k. □

We now prove that the sufficient condition of Lemma 4.1 is also necessary: if at least a *player* does not become *suspended*, the Iteration phase does not terminate.

Lemma 4.2 *If at step k there is at least one* accumulator *in the ring, then all the* suspended *entities receive the <alarm> bit at most n time units after becoming* suspended.

Proof. Consider an entity i that becomes *suspended* at time t_1. To become *suspended* i must have completed the waiting at time $t_0 \leq t_1$ without receiving any message, sent a <starting> bit, become *player*, and received at time t_1 a <starting> bit equal to 0 (Rules (1), (2)). Consider an *accumulator* j; it must have received in state *player* a <starting> bit equal to 1 and sent an <alarm> bit (Rule (2)); this has happened at time $t_2 \leq t_0 + d(i,j) \leq t_1 + d(j,i)$. Thus, an <alarm> bit will be received by i at time

$$t_3 \leq t_2 + d(i,j) \leq t_1 + d(i,j) + d(j,i) = t_1 + n.$$

□

The following lemma shows that, at each step, all the entities which become *players* hold the same value.

Lemma 4.3 *At each step k*

a) all waiting *entities holding x^k_{min} start the step with bounded delay; furthermore they are the only entities which become* player*;*

b) all the other waiting *entities become either* head *or* pipeline.

Proof. By induction on k.

Basis. In the first step, because of the wake up process, all entities start with bounded delay. Since $f(n,x) = 2 \cdot n \cdot x$, the entities with the minimum value finish waiting and send the notification message before receiving any message; this is a basic property of the waiting protocol (e.g, [16]). Hence, they become *player*. Moreover all the other *waiting* entities receive a <starting> bit while still waiting; thus, they become *head* or *pipeline* (Rule (1)).

Inductive step. Let the lemma hold for the first k steps. We will show that, if the Iteration phase does not terminate at step k, the lemma holds for step $k+1$. By induction hypothesis, at step k, the *waiting* entities holding the minimum value x^k_{min} start waiting with bounded delay; that is, for any two *waiting* entities i and j, $T^k_i - T^k_j \leq d(j,i)$. Furthermore they become *player* while all the other *waiting* entities become *head* or *pipeline*.

If the algorithm does not stop at step k, then (by Lemmas 4.1 and 4.2) at least one *player* i must become *accumulator* in this step and *waiting* at step $k+1$. In step k, entity i received a <starting> bit δ_i^k time units after it finished waiting; that is, at time $T_i^k + f(n, \Delta^k) + \delta_i^k$. This bit was sent by its neighboring *player* i' at time $T_{i'}^k + f(n, \Delta^k)$; thus

$$T_i^k + \delta_i^k = T_{i'}^k + d(i, i').$$

Notice that i does not know $d(i, i')$, nor $T_{i'}^k$, nor T_i^k; it does however know that

$$\delta_i^k = T_{i'}^k - T_i^k + d(i, i') \leq 2 \cdot d(i, i') \leq 2 \cdot (n-1).$$

It finishes the accumulation of $x_{M_i}^k$ at time $T_i^k + f(n, \Delta^k) + \delta_i^k + x_{M_i}^k$ and becomes *delaying*. It waits an additional amount of $2 \cdot (n-1) - \delta_i^k$ time units and it starts step $k+1$ becoming *waiting* (Rule (4)); that is,

$$T_i^{k+1} = T_i^k + f(n, \Delta^k) + x_{M_i}^k + 2 \cdot (n-1) \tag{1}$$

If i is the only *accumulator*, the lemma trivially holds. Let also j become *accumulator* at step k; then

$$T_j^{k+1} = T_j^k + f(n, \Delta^k) + x_{M_j}^k + 2 \cdot (n-1) \tag{2}$$

Suppose now that the values accumulated by i and j at step k are minima among the *waiting* entities at step $k+1$; that is, $x_{M_i}^k = x_{M_j}^k = x_{min}^{k+1}$. Then $T_i^{k+1} - T_j^{k+1} = T_i^k - T_j^k \leq d(j, i)$ proving *a)*.

Consider now the case in which $x_{M_j}^k > x_{M_i}^k = x_{min}^{k+1}$. Then, in step $k+1$, entity i finishes waiting and sends a message which would reach entity j at time

$$T_i^{k+1} + f(n, \Delta^{k+1}) + d(i, j). \tag{3}$$

The waiting of j ends at time

$$T_j^{k+1} + f(n, x_{M_j}^k - x_{min}^k) \geq T_j^{k+1} + f(n, x_{min}^{k+1} + 1 - x_{min}^k) = T_j^{k+1} + f(n, \Delta^{k+1}) + 2 \cdot n. \tag{4}$$

From 1 and 2, it follows that

$$T_i^{k+1} - T_j^{k+1} = T_i^k - T_j^k + x_{M_i}^k - x_{M_j}^k \leq d(j, i) < n$$

Hence, expression (3) is less than (4); that is, the message sent by i would reach j while it is still waiting, proving *b)*. □

The next lemma shows that the values held by the *players* are increasing at each step.

Lemma 4.4 *If the Iteration phase does not terminate at step k, then $x_{min}^{k+1} > x_{min}^k$.*

Proof. Let the Iteration phase not terminate at step k. Then, by Lemmas 4.1 and 4.2, at least one entity i becomes *accumulator*, accumulates the maximum value $x^k_{M_i} > x^k_{min}$ in the segment r^k_i, becomes *waiting*, and starts step $k+1$. Therefore, the value held by each *waiting* entity i at step $k+1$ is $x^k_{M_i} > x^k_{min}$. That is, $x^{k+1}_{min} > x^k_{min}$. □

The following lemma proves that the maximum value x_M in the ring is not lost during the execution of the algorithm.

Lemma 4.5 *At the beginning of each step, x_M is held by an active entity.*

Proof. Initially, it is trivially true. Consider a step $k \geq 1$, and assume that the Iteration phase does not terminate at this step. Then, by Lemmas 4.1 and 4.2, there is at least an *accumulator*. Each *accumulator* i starts accumulating the largest value in r^k_i. If it completes the accumulation, i becomes *delaying* and then either *waiting*, or *head* or *pipeline* (Rule (4)); thus, if x_M was in r^k_i, it is now held by an active entity. Instead, if while accumulating, i receives a <starting> bit for the $k+1$ step, it becomes *passive*. In this case, in r^k_i, the *head* and all the entities still *pipeline*, when reached by the bit, enter directly stage $k+1$ in state *head* or *pipeline* (Rules (5), (6)). Thus, if x_M was in r^k_i, it will still be held by an active entity at the beginning of step $k+1$. □

Now we are able to show that the Iteration phase correctly terminates. We will prove that, after a finite number of steps, the phase terminates and that, at the end of such a step, the entities can be only in the state *done* or *passive*. Furthermore *done* entities know the maximum value x_M.

Theorem 4.6 *After a finite number K of steps the Iteration phase terminates. Furthermore, at step K:*

a) $x^K_{min} = x_M$;

b) all the players *become* done *and all the other entities are* passive.

Proof. From Lemmas 4.4 and 4.5, it follows that eventually x_M will be held by *players*; that is, there exists a K such that $x^K_{min} = x_M$. Since x_M is the maximum value, then all the *waiting* entities of step K hold the same value. Thus all of them become *player*. Furthermore, neither *pipeline*, nor *head* entities can be in the ring, as this would imply that there are entities holding values greater than x_M. Therefore, each *player* receives a <starting> bit from the neighboring *player* equal to 0, and becomes *suspended*. By Lemmas 4.1 and 4.2, it follows that the algorithm terminates at step K. Furthermore, since no *player* becomes *accumulator*, no <alarm> bit will be sent in the ring, and each *suspended* entity becomes *done* within an additional n time units. □

We will now calculate the complexity of the algorithm.

Theorem 4.7 *The algorithm determines x_M in $O(n \cdot \log m_{min})$ bits and $O(n \cdot x_M)$ time, where m_{min} is the multiplicity of x_{min}.*

Proof. Let us consider first the number of steps required by the Iteration phase in the worst case. It is easy to see that the number of *players* at each step is at least halved. This is due to the fact that if the segment between two *players* at step k does not contain an active entity, then the *player* at the end of the segment receives a starting bit $= 0$, and becomes *suspended*; thus it cannot be a *player* at step $k+1$. Since in the first step only the entities with $x_i = x_{min}$ become *players*, it follows that the maximum number K of steps is $\log m_{min}$.

At each step k, $O(n)$ bits are required to find the minimum x^k_{min} with the waiting technique, and $O(n)$ bits are required to accumulate the maxima with the pipeline technique and the Two-Bit communicators. The notification phase would add another $O(n)$ bits. Thus, the total bit complexity is $O(n \cdot \log m_{min})$.

Let us consider an entity i that becomes *done* at the final step K. Let Λ be the time at which i becomes *done*; then

$$\Lambda \leq T_i^K + f(n, \Delta^K) + 2 \cdot n$$

Since, for $2 \leq k \leq K$,

$$T_i^k = T_i^{k-1} + f(n, \Delta^{k-1}) + x^k_{min} + 2 \cdot (n-1)$$

solving the recurrence and considering that $T_i^1 \leq n-1$, $\sum_{i=1}^{K} x^i_{min} \leq K \cdot x_M$, and $K \leq \log m_{min}$, it follows that

$$\Lambda \leq x_M \cdot (2 \cdot n + \log m_{min}) + (n-1) \cdot (2 \cdot \log m_{min} - 1) + 2 \cdot n$$

The Notification phase requires $O(n + x_M)$ time in the worst case. Hence the total amount of time required by the algorithm is $O(n \cdot x_M)$ $\square$

Corollary If X is a set, the algorithm determines x_M in $O(n)$ bits and $O(n \cdot x_{min} + x_M)$ time.

In other words, if the values are distinct, the algorithm achieves the same bits-time bounds than the solutions for sets, but without requiring a priori knowledge that X is a set.

5 Refinements and Proof of Main Theorem

5.1 Reducing the Time Complexity

In this section, we show how to reduce the time complexity of the algorithm described in the previous section, without increasing its bit complexity.

The basic idea is to use the previous technique not to determine the maximum value x_M, but to find an upper bound $y \geq x_M$ and then solve the simpler problem of finding the maxima in a bounded multiset where the bound is known.

Given an invertible function $f : \mathbf{R} \mapsto \mathbf{R}$ such that

$$f^{-1}(\lceil f(x) \rceil) \geq x, \tag{5}$$

a maxima finding algorithm can be constructed in two phases:

Phase 1 - Consider the multiset $F = \{f_0, f_1, \ldots, f_{n-1}\}$ where $f_i = \lceil f(x_i) \rceil$; clearly, the maximum value in this multiset is $\lceil f(x_M) \rceil$. Solve $Max\langle F; \mathbf{Z}^+ \rangle$ to find $\lceil f(x_M) \rceil$. Since $y = f^{-1}(\lceil f(x_M) \rceil) \geq x_M$, an upper bound on the maximum of the original value is now available.

Phase 2 - Consider the multiset $Y = \{y_0, y_1, \ldots, y_{n-1}\}$, where $y_i = y - x_i$ or $y_i = *$ depending on whether or not $f_i = f_{max}$, respectively; clearly, the minimum value (different from $*$) in Y is $y_{min} = y - x_M$. Solve $Min\langle Y; \mathbf{Z}^+ \cup \{*\} \rangle$ to find y_{min}, where the entities whose value is $*$ act as relayers.

Using the algorithm described in Section 3, Phase 1 can be completed in $O(n \cdot f(x_M))$ time with $O(n \cdot \log n)$ bits. Using the Waiting technique (e.g., [16]), Phase 2 requires $O(n)$ bits and $O(n \cdot y_{min})$ time. Thus, the total bit complexity of finding the maxima using this two-phase algorithm is $O(n \cdot \log n)$, while the time complexity is $O(n \cdot Max\{f(x_M), f^{-1}(\lceil f(x_M) \rceil) - x_M)\})$.

For example, using the function $f(x) = \log x$, the bit complexity is $O(n \cdot \log n)$; the time complexity becomes $O(n \cdot Max\{\log x_M, (2^{\lceil \log x_M \rceil} - x_M)\})$ which, for some values of x_M, can be less than $O(n \cdot x_M)$.

The choice of f directly influences the time complexity whereas the bit complexity is $O(n \log n)$ regardless of the choice. To minimize the time complexity, f must be such that

$$f^{-1}(\lceil f(x) \rceil) - x = O(f(x)) \tag{6}$$

Observe that, for the choice $f(x) = \log x$ described above, (6) does not hold.

A function that satisfies both (5) and (6) is $f(x) = \sqrt{x}$, since $\lceil \sqrt{x} \rceil^2 - x \leq 2\sqrt{x}$. For such a choice, we obtain a maximum finding algorithm which requires $O(n \cdot \log n)$ bits and $O(n \cdot \sqrt{x}_M)$ time, improving the bounds obtained in Section 3. It can be shown that any other choice of f would not lead to further improvements in the order of magnitude of the time complexity.

This approach has a major drawback. The bit complexity of the algorithm of Section 3 is sensitive to the multiplicity of the minimum value in X. In particular, if X is a set, the algorithm requires only $O(c \cdot n)$ bits, even if this information is not known a priori. Because of the "ceiling" operator, F might be a multiset even when X is a set (in the worst case, all f_i's could be equal). Thus, the overall bit complexity could increase with the proposed approach.

This drawback can be easily solved. In fact, to maintain the sensitivity of the original algorithm, it is sufficient to add a **Phase 0** to the strategy; in this phase, the minimum value in X will be found and it will be detected whether it is unique. Using the algorithm of [10], the minimum value can be found with $O(c \cdot n)$ bits and $O(c \cdot n \cdot x_{min}^{\frac{1}{c}})$ time for any integer $c > 0$; determining whether it is unique requires an additional $O(n)$ time and bits. Therefore, for any $c > 1$, the complexity of Phase 0 is not greater than the one of the other two phases. In other words, we can reduce the time complexity of the algorithm of Section 3 maintaining the sensitivity to the distinctness of the minimum value, and without increasing the bit complexity.

5.2 Bits-Time Tradeoff and Main Theorem

In the previous section, we have reduced the time complexity by first finding an upper bound y on the x_M and then determining the minimum of the values $y - x_i$. We will now show how to extend this approach to derive a continuum of bits-time tradeoffs, which will lead to the proof of the Main Theorem. Namely, we will prove that, for any a priori choice of an integer $c > 0$, it is possible to find the maxima in $O((c + \log n) \cdot n)$ bits and $O(c \cdot n \cdot x_M^{\frac{1}{c}})$ time. If the values are distinct, the bit complexity is $O(c \cdot n)$.

The basic idea is as follows. As before, we first find an upper bound y on the maximum x_M using the technique described in the previous section and consider the multiset $\{y_0, \ldots y_{n-1}\}$, where $y_i = y - x_i$. To determine x_M, we decompose each value y_i into a c-tuple $\langle y'_{i,1}, y'_{i,2}, \ldots, y'_{i,c} \rangle$, with values in $\mathbf{Z}^+ \cup \{*\}$ and apply the minimum finding algorithm (waiting technique) to the multisets $Y_t = \{y'_{0,t}, y'_{1,t}, \ldots, y'_{n-1,t}\}$, $1 \leq t \leq c$, sequentially until $y_{min} = y - x_M$ is determined.

Specifically, to find the upper bound, we use the function $f(x) = \log x$; thus we have $y = 2^{\lceil \log x_M \rceil}$ and $y_i = 2^{\lceil \log x_M \rceil} - x_i$.

The values $y'_{i,t} \in \mathbf{Z}^+ \cup \{*\}$ are chosen so that:

1) if $y'_{i,t} \in \mathbf{Z}^+$ then $y'_{i,t} = O(x_i^{\frac{1}{c}})$;
2) if $y_i = y_{min}$, then $\forall j \; y'_{i,t} \leq y'_{j,t}$ (where every integer is assumed to be less than $*$);
3) if $y_j < y_i = y_{min}$, then there exists a t such that $y'_{j,t} > y'_{i,t}$.

A preliminary phase is executed to make the bit complexity sensitive to the multiplicity of the minimum value.

Phase 0 - Solve $Min\langle X; \mathbf{Z}^+ \rangle$ to find x_{min}. If the minumum is unique, the corresponding entity determines the maximum value and the algorithm terminates; otherwise, Phase 1 is executed.

Phase 1 - Consider the multiset $F = \{f_0, f_1, \ldots, f_{n-1}\}$ where $f_i = \lceil \log(x_i) \rceil$; clearly, the maximum value in this multiset is $\lceil \log(x_M) \rceil$. Solve $Max\langle F; \mathbf{Z}^+ \rangle$ to find $\lceil \log(x_M) \rceil$.

Since $y = f^{-1}(\lceil f(x_M) \rceil) = 2^{\lceil \log x_M \rceil} \geq x_M$, an upper bound y on the maximum of the original value is now available.

Consider the multiset $\{y_0, \ldots, y_{n-1}\}$, where $y_i = y - x_i$.

Let $y_{i,t}$ and $z_{i,t}$, $0 \leq i \leq n-1$ and $1 \leq t \leq c$, be recursively defined as follows:

$$y_{i,t} = \lfloor z_{i,t}^{\frac{1}{c-t+1}} \rfloor \tag{7}$$

where

$$z_{i,t} = z_{i,t-1} - y_{i,t-1}^{c-t+2} \tag{8}$$

and $z_{i,1} = y_i$ and $y_{i,1} = \lfloor y_i^{\frac{1}{c}} \rfloor$.

Phase 2 - Consists of at most c steps:
step 1: solve $Min\langle Y_1; \mathbf{Z}^+ \rangle$ where $Y_1 = \{y'_{0,1}, \ldots, y'_{n-1,1}\}$ and $y'_{i,1} = y_{i,1}$.
If the minimum value $y'_{min,1}$ is unique, terminate the algorithm; otherwise proceed.
step t ($2 \leq t \leq c$): solve $Min\langle Y_t; \mathbf{Z}^+ \rangle$, where $Y_t = \{y'_{0,t}, y'_{1,t}, \ldots, y'_{n-1,t}\}$ and:

$$y'_{i,t} = \begin{cases} y_{i,t} & \text{if} \quad y'_{i,t-1} = y_{min,t-1} \\ * & \quad \text{otherwise} \end{cases}$$

(Every entity with value $*$ acts only as a relayer). If the minimum $y'_{min,t}$ is unique or $t = c$, terminate the algorithm, otherwise proceed to step $t+1$.

Theorem 5.1 *The maximum x_M can be found in $O((c + \log n) \cdot n)$ bits and $O(c \cdot n \cdot x_M{}^{\frac{1}{c}})$ time, for every integer $c > 0$. If the values are distinct, the bit complexity is $O(c \cdot n)$ and the time complexity is $O(c \cdot n \cdot x_{min}^{\frac{1}{c}})$.*

Proof. Consider the three-phase algorithm described above. If the minimum value is unique, this situation is detected in Phase 0, the maximum value is found, and the algorithm terminates. Otherwise, the algorithm procedes with Phase 1. Using the algorithm of [10], the bit complexity of Phase 0 is $O(c \cdot n)$ and the time complexity is $O(c \cdot n \cdot x_{min}^{\frac{1}{c}})$.

In Phase 1 the upper bound can be found in $O(n \cdot \log n)$ bits and time (see Section 5.1).

Phase 2 is composed of at most c steps; step t, $1 \leq t \leq c$, is an execution of the waiting technique on the multiset Y_t.

In each step t, there are two types of entities: relayers (i.e., $y'_{i,t} = *$) and players (i.e., $y'_{i,t} \in \mathbf{Z}^+$). Initially all entities are players. By construction, if $y'_{i,t} = *$ then $y'_{i,t+1} = *$; that is if a player i becomes a relayer, it will remain so until the end of the algorithm.

If entity i is a player in step t, then $y'_{i,t} = y_{i,t}$. By definitions (3) and (4)

$$y_{i,t} = \lfloor (y_i - \sum_{k=1}^{t-1} y_{i,k}^{c-k+1})^{\frac{1}{c-t}} \rfloor$$

Hence, if $y_i = y_{min}$, then $\forall j$ and $\forall t$, $y'_{i,t} \leq y'_{j,t}$ (where every integer is assumed to be less than $*$). Furthermore, if $y_j > y_i = x_{min}$, then there exists a $t \leq c$ such that $y'_{j,t} > y'_{i,t}$. Thus, entity j becomes a relayer at step t. Therefore, when the algorithm terminates, all and only the entities whose value is y_{min} are in state *minimum*.

Step t is an execution of the waiting technique on the multiset Y_t; thus it requires $O(n)$ bits and $O(n \cdot y_{min,t})$ time, plus an additional $O(n)$ bits and time to determine whether $y_{min,t}$ is unique.

We will now show that $z_{min,t+1} = O(x_M^{\frac{c-t+2}{c}})$ and, thus, $y_{min,t+1} = O(x_M{}^{\frac{1}{c}})$. For $t = 1$ it is true, in fact

$$z_{m,1} = y_m = 2^{\lceil \log x_m \rceil} - x_M = O(x_M)$$

Let it hold for step $t < c$; that is $z_{i,t} = O(x_i^{\frac{c-t+1}{c}})$ and $y_{i,t} = O(x_i{}^{\frac{1}{c}})$. It follows that

$$z_{i,t+1} = z_{i,t} - y_{i,t}^{c-t+1} = O(x_i^{\frac{c-t+1}{c}})$$

Thus, Phase 2 requires at most $O(c \cdot n)$ bits and $O(c \cdot n \cdot x_M{}^{\frac{1}{c}})$ time.

□

By Theorem 5.1 and Theorem 2.1, it follows

Main Theorem *For any integer $c > 0$, EF can be solved with $O((c + log\, n) \cdot n)$ bits in time $O(c \cdot n \cdot x^{\frac{1}{c}})$, where $x = Max\{|x_{min}|, |x_{max}|\}$.*

Notice that the function $c \cdot n \cdot x^{\frac{1}{c}}$ has a single minimum at $c = ln\, x_M$; for such a value the time complexity is $O(n \cdot \log x_M)$ which is less than the one obtained by [2] with bounded values.

6 Finding the Multiplicity of the Extrema

In this section, we consider the *Extrema Counting Problem* (EC); that is, the problem of finding the multiplicity of the extrema in the multiset X. We show how the proposed algorithm for EF directly implies the same bits-time upperbound for EC.

In the following, we will describe only the solution to the problem $MAXC\langle X, \mathbf{Z}^+ \rangle$ of finding the multiplicity of the maxima in X when $U = \mathbf{Z}^+$: as in the case of EF, this technique leads to correct solution of the general case $EC\langle X, \mathbf{Z} \rangle$.

The algorithm is composed of two phases.

First Phase:

> the entities determine the maxima using the algorithm presented in Section 5. At this point, each entity i sets $v_i = 1$ if it is *maximum* (i.e., $x_i = x_M$), and $v_i = 0$ if it is *small* (i.e., $x_i < x_M$). Let $V = \{v_0, v_1, ..., v_{n-1}\}$ be the resulting multiset.

Second Phase:

> the entities solve $SUM\langle V, \{0,1\}\rangle$ (i.e., determine the number of ones in V) using the algorithm of [1].

The first phase requires an execution of the algorithm of Section 5. The second phase of the algorithm requires $O(n \cdot \log n)$ bits and time [1].

In other words:

Theorem 6.1 *For any integer constant $c > 0$, EC can be solved with $O((c + \log n) \cdot n)$ bits and $O(n \cdot max\{x^{1/c}, \log n\})$, where $x = Max\{|x_{min}|, |x_{max}|\}$.*

We will now show that the proposed algorithm is bit optimal. This will be done by "lifting" an existing result for computing the XOR of Boolean values.

Given two problems W_1 and W_2, we will denote by $W_1 \geq_b W_2$ the fact that the bit complexity of W_2 is not greater than that of W_1.

Theorem 6.2 *Any solution to EC requires $\Omega(n \cdot logn)$ bits*

Proof. Given $X = \{x_0, x_1, ..., x_{n-1}\}$, let $V = \{v_0, v_1, ..., v_{n-1}\}$, where $v_i = 1$ iff $x_i = x_M$. Then $EC\langle X, \mathbf{Z}\rangle \geq_b MAXC\langle X, \mathbf{Z}\rangle \geq_b MAXC\langle V, \{0,1\}\rangle \geq_b XOR\langle V, \{0,1\}\rangle =_b \Omega(n \cdot \log n)$; the lower bound on the XOR has been established in [1]. □

Thus, the proposed algorithm has an optimal bit complexity.

7 Extensions and Remarks

7.1 Bidirectional Rings

In the previous section, we have considered unidirectional rings; that is, rings where each entity i can send messages only to $i + 1$ and receive messages only from $i - 1$.

In the case of bidirectional rings, every entity i can directly communicate with both $i+1$ and $i-1$. Recall that the indices i, $i+1$ and $i-1$ are used here only for convenience of notation and are not known to the entities. Rather, each entity is only aware of its two neighbours, locally known as "left" and "right". If the local assignment of these labels is globally consistent (e.g., each i knows $i + 1$ as "right"), the ring is said to be *oriented.*

In oriented bidirectional rings, any unidirectional algorithm can be executed with the same bits-time complexity by using only one direction for all communications. Thus, for such rings, all the results discussed in the previous sections trivially hold.

If no orientation exists but the ring size n is odd, then the ring can be oriented with $O(n \cdot \log n)$ bits and time [2]. Thus, also in this case, the estabilished bounds hold.

Finally consider unoriented rings of even size. These rings cannot be oriented (i.e., no deterministic orientation algorithm exists) [2]. In this case we can still achieve the same bounds as follows. Every entity runs two parallel and independent executions of the unidirectional algorithm, one for each local direction; when a message arrives, the direction of the arrival will determine to which execution it must be applied. In this way, the correctness will be unchanged and the complexity will be at most doubled.

7.2 Concluding Remarks

We have considered the *Extrema-Finding Problem*; that is, the problem of finding the minimum and the maximum values, x_{min} and x_{max}, of a multiset $X = \{x_0, x_2, ..., x_{n-1}\}$, whose elements are drawn from a totally ordered universe U and stored at the n entities of a ring network.

We have shown that, in synchronous rings of known size, this problem can be solved in $O((c + \log n) \cdot n)$ bits and $O(c \cdot n \cdot x^{\frac{1}{c}})$ time for any integer $c > 0$, where $x = Max\{|x_{min}|, |x_{max}|\}$. The previous solutions required $O(n^2)$ bits and the same amount of time.

The Extrema Finding problem is unsolvable if the ring size is unknown to the entities; it has complexity $\Theta(n^2)$ in the case of asynchronous rings of known size. Thus, the only question still open is whether the established bits-time bounds for synchronous rings of known size are tight.

We have also presented presented an $O(n \cdot logn)$ solution to the problem of finding the multiplicity of the extrema, and shown it to be bit optimal. The last result leads us to the *conjecture* that $\Omega(n \cdot logn)$ is also a lower bound on the bit complexity of Extrema Finding and, thus, the proposed solution is bitwise optimal.

References

[1] H. Attiya, M. Snir, M.K. Warmuth, Computing on an Anonymous Ring, in *Proc. of 4th ACM Symposium on Principles of Distributed Computing*, 1985, 196–203.

[2] H. Attiya, M. Snir, M.K. Warmuth, Computing on an Anonymous Ring, *Journal of the ACM*, vol. 35, n. 4, 1988, 845–875.

[3] A. Bar-Noi, J. Naor, M. Naor, One Bit Algorithms, *Distributed Computing*, vol. 4, n. 1, 1990, 3–8.

[4] H.L. Bodlaender, New Lower Bound Techniques for Distributed Leader Finding and Other Problems on Ring Processors, *Theoretical Computer Science*, vol. 81, 1991, 237–256.

[5] H.L. Bodlaender, G. Tel, Bit-optimal Election in Synchronous Rings, in *Information Processing Letters*, vol. 36, n. 1, 1990, 53–56.

[6] J. Burns, A Formal Model for Message Passing Systems, Technical Report UTR-91, Indiana University, 1981.

[7] D. Dolev, M. Klawe, M. Rodeh, An $O(n \log n)$ Unidirectional Algorithm for Extrema-Finding in a Circle, *Journal of Algorithms*, vol. 3, 1986, 245–260.

[8] G.N. Frederickson, N.A. Lynch, Electing a Leader in a Synchronous Ring, *Journal of the ACM*, vol. 34, n. 1, 1987, 95–115.

[9] E. Gafni, Improvements in the Time Complexity of two Message-Optimal Election Algorithms, in *Proc. of 4th ACM Symposium on Principles of Distributed Computing*, 1985, 175–185.

[10] J. van Leeuwen, N. Santoro, J. Urrutia, S. Zaks, Guessing Games and Distributed Computations in Synchronous Networks, in *Proc. of 14th International Colloquium on Automata, Languages and Programming*, vol. 267, 1987, 347-356.

[11] A. Marchetti-Spaccamela, New Protocols for the Election of a Leader in a Ring, *Theoretical Computer Science*, vol. 54, n. 1, 1987, 53–64.

[12] U.M. O'Reilly, N. Santoro, The Expressiveness of Silence: Tight Bounds for Synchronous Communication of Information Using Bits and Silence, in *Proc. of 18th International Workshop on Graph-Theoretic Concepts in Computer Science*, 1992, 321–332.

[13] M. Overmars, N. Santoro, Improved Bounds for Electing a Leader in a Synchronous Ring, *Algorithmica*, to appear.

[14] J. Pachl, D. Rotem, E. Korach, Lower bounds for Distributed Maximum Finding algorithms, *Journal of the ACM* , vol. 31, 1984, 380–401.

[15] G.L. Peterson, An $O(n \log n)$ Unidirectional Algorithm for the Circular Extrema Problem, in *ACM Transaction on Programming, Languages and Systems*, vol. 4, 1982, 758–762.

[16] N. Santoro, D. Rotem, On the Complexity of Distributed Elections in Synchronous Graphs, in *Proc. of 11th International Workshop on Graph-Theoretic Concepts in Computer Science*, 1985, 337–346.

[17] B. Schmeltz, Optimal Tradeoffs between Time and Bit Complexity in Synchronous Rings, in *Proc. of 10th Symposium on Theoretical Aspects of Computer Science*, 1990, 275-284.

[18] P. Spirakis, B. Tampakas, Efficient Distributed Algorithms by using the Archimedean Time Assumption, in *Proc. of 5th Symposium on Theoretical Aspects of Computer Science*, 1988, 248–263.

[19] P. Vitanyi, Distributed Elections in an Archimedean Ring of Entities, in *Proc. of 16th ACM Symposium on Theory of Computing*, 1984, 542–547.

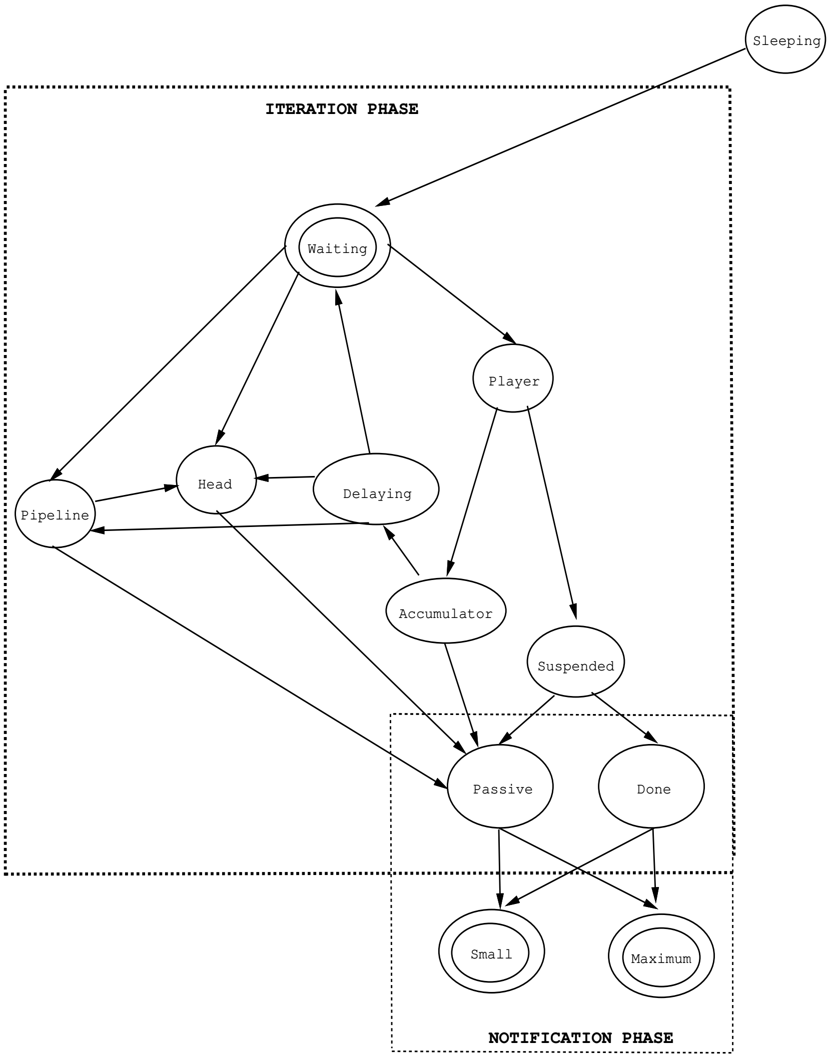

Figure 1: State Transition Diagram