Exact Communication Costs for Consensus and Leader in a Tree

YEFIM DINITZ, *Department of Computer Science, Ben-Gurion University of the Negev, Israel. E-mail: dinitz@cs.bgu.ac.il*

SHLOMO MORAN, *Department of Computer Science, Technion, Israel. E-mail: moran@cs.technion.ac.il*

SERGIO RAJSBAUM, *Compaq Cambridge Research Laboratory, MA. E-mail: Sergio.Rajsbaum@compaq.com. On leave from Instituto de Matemáticas, UNAM, Mexico. rajsbaum@math.unam.mx*

ABSTRACT: We continue the study of communication costs of `Consensus` and `Leader` initiated in a previous paper [7]. We deal with all scenarios with linear complexity in a tree topology, and prove exact (as opposed to asymptotic) tight bounds for the bit and message complexities. A particular scenario depends on whether the tree size or the size parity is known to the processors.

Keywords: Distributed Computing, Consensus, Leader Election, Bit Complexity, Message Complexity, Communication Complexity.

1 Introduction

The two most studied tasks in the distributed computing literature are probably `Consensus` and `Leader Election`. These tasks can be viewed as duals in the sense that in `Consensus` all processors have to agree on the same bit, while in `Leader` one processor has to output a different value from the rest of the processors. In this sense, `Consensus` is about achieving symmetry, while `Leader` is about breaking symmetry. A large number of results about these tasks exist, including algorithms, lower bounds and applications, in a variety of distributed computing models (see [1, 17, 19]). However, `Consensus` and `Leader` have been considered in different circumstances. Typically, `Consensus` is very important in fault-tolerance studies; a lot of research on this subject was done following the pioneering work of [9]. On the other hand, `Leader` is the main paradigm for message complexity studies, see eg [4, 12].

In this paper[1] we continue an in-depth study of the relation between the communication costs of `Consensus` and `Leader`in the fault-free model, started in our previous

[1] A preliminary version of this paper was published in [8].

 © Hermes Science Publications

paper [7]. The model is the standard asynchronous, failure-free message passing distributed system, with arbitrary but finite link delays and negligible local computation times. We study networks which are either an arbitrary tree or a chain. There are n links in the network and the processors get different identifiers ("ids", for short) from the set $Z_M = \{0, \ldots, M-1\}$.

Our interest in the relation between the communication costs of `Consensus` and of `Leader` started by noticing that both tasks have the same asymptotic, worst case message complexity in all standard topologies (the lower bound for `Consensus` in rings was proved in [7] for n unknown, and in [8] for n known. In contrast, `Leader` is harder than `Consensus` in a ring in the following sense [7]). Once a leader has been elected, the leader can broadcast a decision value, and thus solve `Consensus` with $O(n)$ messages. On the other hand, even if `Consensus` has been solved, $\Theta(n \log n)$ messages are still needed to solve `Leader`.

The case of a chain turns out to be particularly interesting. The traditional message complexity measure does not distinguish between `Consensus` and `Leader` in a chain: it is easy to see that to solve both, $\Theta(n)$ messages are necessary and sufficient. In [7], a refined measure, *bit complexity*, was used to distinguish between the communication costs of these tasks (see also the relation to communication complexity theory described below).

Our Results. In this paper we show that `Consensus` and `Leader` are not equally difficult in a chain, by proving exact (instead of asymptotic) bounds for message and bit complexities. Moreover, we show how to extend all of our upper bounds to a tree topology.

We deal with all chain scenarios where `Consensus` and `Leader` have *linear* complexities. A particular scenario depends on whether the length of the chain, or its parity, is known to the processors, and whether ids are at all processors or at the terminals (ie, the chain's end nodes) only. The only scenario that gives raise to a non $O(n)$ complexity (depending also on M), is when the length of the chain is odd and the task is `Leader`. This scenario is where the main asymptotic bit complexity results of [7] were proved. The most interesting case is when ids are given to the two terminal processors only; the complexity is $\Omega(n \log M)$.

Our results for the tree topology are summarized in Figure 1. The upper bounds for `Consensus` hold for both trees with ids only at the terminals (i.e., the tree's leaves) and trees with ids at every processor. For `Leader`, in the case where processors know n, the message and bit complexity lower bounds hold even if all processors have ids, while the upper bounds hold even if only the terminals have ids. All our lower bounds are proved for a chain, and thus hold for a general tree (since a chain is a particular kind of tree).

It is quite surprising that a leader in a tree of even n can be elected (that is, an undirected tree can be converted into a rooted directed one) with only $1.5n$ bits, i.e., without sending one bit per direction of every link. We show that this bound is tight. We use the same lower bound proof strategy to prove the same lower bound for odd n (but note that in [7] we prove the stronger bit complexity lower bound of $\Omega(n \log M)$ for odd n).

Our results show that, for a tree, the communication cost of `Consensus` does not depend on any information the processors might have about n. In contrast, the message and (in some cases) bit complexity of `Leader` is sensitive to this knowledge. In other words, the complexity of breaking symmetry depends on the knowledge that processors have on the network, while that of achieving symmetry does not.

Problem	Knowledge on n	Lower Bound	Upper Bound
`Consensus`	all cases	$2n$ msgs	$2n$ msgs
`Leader`	n known	$n+1$ msgs	$n+1$ msgs
`Leader`	n unknown	$2n$ msgs	$2n$ msgs
`Consensus`	all cases	$2n$ bits	$2n$ bits
`Leader`	unknown even n	$1.5n$ bits	$1.5n$ bits

FIG. 1. Our results for the tree topology.

Related Work. Our study of the relation between the communication costs of `Consensus` and `Leader` was started in [7] using the bit complexity measure. In particular, we discovered in [7] that the bit complexity of `Consensus` in a chain does not depend on M, while the bit complexity of `Leader` in most cases does depend on M and is sensitive to the information known about the network. Figure 2 presents the results of [7]. It includes the task of electing as the leader the processor with largest identifier, `MaxF`. (The algorithm for `Leader` in the even chain case does not use the ids. In the case of `MaxF`, at the bottom of the figure, the lower bound for a chain holds even under the best possible scheduler.)

Problem	Variant	Lower Bound	Upper Bound
`Consensus`	$n=2$	2	2
`Leader`, `MaxF`	$n=2$	$2\lfloor \log M \rfloor - 3$	$2\lceil \log M \rceil - 2$
`Consensus`	chain	$\Omega(n)$	$2n$
`Leader`	even chain	$\Omega(n)$	$2n$
`Leader`	odd chain, ids at all	$\Omega(n+\log M)$	$2(n+\log M)$
`Leader`	odd chain, ids at ends	$\Omega(n\log M)$	$2n\log M$
`MaxF`	chain	$\Omega(n\log M)$	$2n\log M$

FIG. 2. Previous bit complexity results (parities mentioned are of n).

Our bit complexity questions are related to *communication complexity* (see [16]). The basic problem in communication complexity, introduced by Yao [21], asks what is the number of bits that two parties, Alice and Bob, have to communicate to each other in order to compute a function $f(x,y)$ of their respective, private inputs, x,y. The model assumes that they send bits one at a time, starting with one or more bits sent by Alice, then some bits sent by Bob, and so on. A large number of results exist on this problem and its variants, which consider also more general networks. In particular, the chain topology has attracted attention. In [5, 15, 20] the relation between the two processor communication complexity of a function, and the communication complexity of the same function on a chain was studied.

From the point of view of communication complexity, our setting introduces new difficulties to the problem. First, processors are not distinguished a priori. In particular, they do not have commonly known identities, and there is no fixed order of sending messages that might distinguish them (e.g. always one of them starts the computation). Also, we are interested in problems specified by a *task*: an input/output relation where several outputs are allowed for the same input. In addition, the output does not have to be the same for all processors. These are the usual assumptions in the distributed computing literature (e.g. [1, 17, 19]).

Organization. This paper is organized as follows. In Section 2 we describe the model of computation, the notion of a scheduler and a particular scheduler used in the rest of the paper, and some definitions about trees. In Section 3 we present our bit and message complexity results for `Consensus`. Section 4 presents the message complexity results for `Leader`, and Section 5 presents the bit complexity results for it. Section 6 contains some concluding discussion.

2 Preliminaries

In Section 2.1 we describe the model of computation, which is the standard asynchronous, failure-free message passing model. In Section 2.2 we describe our notion of scheduler, and a particular scheduler which is useful later on. In Section 2.3 some definitions about trees are presented. In Section 2.4 we describe a general strategy to compute a function in a tree, on which all of our algorithms are based.

2.1 Model

In this section we briefly describe the model of computation and the distributed problems we are interested in. More precise descriptions can be found in textbooks such as [1, 17, 19]. We consider a standard asynchronous, failure-free message-passing distributed system, consisting of n links connecting pairs of processors. The processors are arranged in a tree topology. The processors have distinct identifiers (ids) from the set $Z_M = \{0, \ldots, M-1\}$.

A *distributed algorithm* consists of a set of identical sequential, deterministic *local* algorithms, one for each processor. The algorithms of the processors are identical in the sense that they depend only on the processors identities and number of incident links. A local algorithm includes instructions for sending and receiving messages. All processors are in the same initial state, except for the id and incident link information. Thus, a processor does not know the ids of other processors.

When a processor wakes up spontaneously, or when it receives one or more messages, it makes some internal computations according to its local algorithm, and then possibly sends messages on incident links. *Any processor* can wake up spontaneously.

In our algorithms for the tree topology, we assume that *all terminals except for at most one* eventually wake up spontaneously, if have not received a message yet. We use this assumption to allow a processor with m links to wait for a message on $m - 1$ of its links before sending a message. See Section 6 for a discussion of this assumption. In our lower bound proofs, we use a weaker assumption that *all terminals* which have not received a message yet, eventually wake up spontaneously.

We assume that the internal computation time is negligible, which implies that messages are sent by a processor only when it wakes up or as an immediate response to receiving a message; we say that the processor is *active* at this moment. Messages are delivered after a finite but otherwise arbitrary delay. Our lower bounds hold even if the messages are delivered in FIFO order, while our algorithms do not require this assumption.

In a *decision task*, the processors start with input values and must eventually decide on output values which satisfy the task's specification. In this paper, the inputs are the processors id's, and hence any vector of distinct integers from Z_M, one for each processor, is a possible input to the system. We also consider scenarios where only the terminals have id's (ie, the remaining processors are anonymous). The outputs are binary.

We are interested in the complexities of the following tasks:

1. `Consensus`: all processors must decide on the same bit. If all the id's are odd they must output 1, if all id's are even they must output 0.
2. `Leader`: one processor decides 1, the rest decide 0.

For `Consensus` we refer to the least significant bit of a processor id as to its *input bit.* For `Leader` in a tree, we require also that each non-leader processor knows which of its incident links is directed towards the leader.

We consider the following complexity measures:

- *Message Complexity*: number of messages sent, denoted MsgC.
- *Bit Complexity*: number of bits sent, denoted BitC.

We consider *worst case* complexities, in which the above figures are maximized over all input configurations and all possible executions.

Remark. Our processors do not have sense of direction [10], an assumption that would be too strong for our problems. For instance, in a chain with sense of direction `Leader` is solved with no communication at all (the leader is the terminal processor on the "left," say), while `Consensus` can be solved with only n bits (the leader broadcasts its bit).

2.2 Schedulers

Given a distributed system, a *scheduler* defines the order in which processors wake up spontaneously and the order in which messages are delivered. The next configuration is determined by the scheduler using the following two steps process:

- The scheduler wakes up a processor, and/or delivers one or several messages to one processor.

- The processor activated by the scheduler above, executes in negligible time a step, which consists of making local computations and possibly sending some messages, according to its local program.

An *execution* is defined by a given scheduler as follows. Initially, all processors are asleep; each next configuration is determined by the current configuration and the decisions made by the scheduler as above. The worst case time is maximized over all possible schedulers. Clearly, any lower bound for an arbitrary algorithm and a *fixed* scheduler is a general lower bound.

For lower bound proofs for the chain topology, we use the following *backward-preference* scheduler (a variant of the scheduler introduced in [7] by the name "outside-precedence scheduler"). Suppose, a chain terminal, P, has waken up spontaneously. Consider the messages sent by it, the processors waken up by these messages, the messages sent from these processors, and so on. Let us call such a process the *wave* generated by P. In each execution there can be two waves (one from each terminal) executed simultaneously. The duration of a wave is up to the moment when (1) a message sent in a wave reaches the other terminal or a processor covered by the other wave (a *wave meeting*), or when (2) the wave comes to a *standstill*, or *stops*: there are no pending messages to be delivered, in this wave.

The backward-preference scheduler does not wake up non-terminals. It delivers messages sent by a wave originated at terminal P as follows:

- it delivers one message at a time; this message is always an undelivered message on the link which is closest to P, and
- it delivers a message to a processor covered by the other wave only when there are no undelivered messages also between processors in that second wave.

This implies that a message on some link e is delivered only when there are no undelivered messages between P and e. Note that the backward-preference scheduler is well defined for two waves executed simultaneously: indeed, before the wave meeting the waves are completely independent. Suppose that two waves, under the backward-preference scheduler, meet, and both are not at a standstill. It is easy to see that just before the wave meeting, there are only two messages in the system, and they are in transit at the same link, in opposite directions.

Remark. The backward-preference scheduler facilitates our lower bound proofs, by forcing a simple and uniform behavior of an *arbitrary* algorithm. Specifically, it forces any algorithm to behave under the following expansion strategy: The set of processors already covered by the wave behave like a single super-processor which expands its area step by step, with "internal" computation steps executed completely between any two such consecutive expansion steps.

2.3 *Trees*

We assume the reader is familiar with the standard notations of trees and directed trees. For our analysis, we need the following notation: Let $e = (P, Q)$ be any link in a tree T with $N = n + 1$ processors. Removal of e divides T into two parts, one containing

P and the other containing Q. The part of T which contains Q is the *subtree hanging on e from P*, and Q is its *root*. When link e is distinguished, among the links incident to Q, then the *subtree rooted at Q* denotes the subtree hanging on e and rooted at Q.

We call the number of processors in a tree or subtree its *weight*. A processor P of T is called its *center* if the weight of all subtrees hanging from it is at most $N/2$. It is well known that, in a tree, either there is exactly one center, and then all subtrees hanging on it have the weight strictly less than $N/2$, or there is a *central link* $e = (P, Q)$ such that both subtrees hanging on e and rooted at P and Q have weight $N/2$, and then there are two centers: P and Q (this case is possible only for odd n).

2.4 Collection of Information Strategy

In the algorithms of this paper we use variants of a strategy to propagate information from the terminal processors to the rest of the tree and back. This strategy, `CIF` (Collection of Information with Feedback), consists of two phases. We also use a version of this strategy, `CI`, that executes only the first phase.

The `CIF` strategy is as follows. First, in the *collect phase,* when a terminal processor wakes up, it sends a message to its neighbor, which is its *parent* in this execution (in the applications this message will contain information about its id). An internal processor P with m incident links ignores a wake-up by the scheduler, if any, and waits until it receives messages on exactly $m - 1$ links; then it sends a message on its remaining link (in the applications this message will contain information about the number of processors and their ids in the $m - 1$ subtrees hanging from P on the above links). The processor on the other end of this link is P's parent, and its other neighbors are P's *children* in the execution. Then, in the *feedback phase*, any processor waits for a message from its parent, and then sends a message to each of its children, if any. A processor terminates when it finishes executing the feedback phase. It is easy to see that the algorithm always terminates, due to the assumption that all terminals except for at most one, wake up spontaneously.

Let $f : V \times V \to V$ be a commutative and associative function defined on a (finite) set V; then f can be naturally extended to define a function from all multisets containing at least two (not necessarily distinct) elements of V into V. We also define $f(v) = v$ for each $v \in V$. Such a function f can be the size of a multiset (number of inputs), minimum, maximum, gcd, etc. The `CIF` algorithm can be used to compute any such function f of the input values at all processors in a tree, so that upon termination, every processor knows the value of f. We extend this definition to the case when only a part of the processors in a tree have inputs, by assuming that the empty value $\perp$ ($\perp \notin V$) is given to a processor if it has no input, assuming $f(v, \perp) = f(v)$ for every $v \in V$. In our applications, inputs are given either to all processors, or to the terminals, and V can be the set of ids, or the multiset of parities of ids.

THEOREM 2.1
The `CIF` algorithm can compute, at each processor, any associative and commutative function on the multiset of input values. This is done by $2n$ messages, each one with at most $\lceil \log_2 |V| \rceil$ bits.

PROOF. Let I be the multiset of input values, and f the function to be computed. Each processor maintains a value v. Initially v contains its input, or an empty value $\perp$, if no input is given to it. Each time a processor receives a message with a value u, it computes $v \leftarrow f(v, u)$. Each message sent by a processor consists of its current value v.

We show that at any moment during the execution of the algorithm, the value of v at a processor P satisfies $v = f(U)$, where U is the set of input values in the subtree rooted at P formed by P and the subtrees hanging from P on all incident links on which P has received a message. The proof is by induction on the number of steps in the execution. Initially the claim holds. Consider any global state in an execution of the algorithm, and assume the claim holds at every processor. Assume P has value v_P in this global state, and receives a message on e from Q with value v_Q, so it executes $v_P \leftarrow f(v_P, v_Q)$. By the induction hypothesis, v_Q is equal to $f(U_Q)$, where U_Q is the set of input values in a subtree rooted at Q. Moreover, since Q sends a message on a link e only when it has received messages on all other links, this subtree is the *entire* subtree rooted at Q hanging on e from P. It follows that when P receives this message, the value it computes $v_P \leftarrow f(v_P, v_Q)$ is the correct value. ∎

The `CIF` strategy is a generalization of the distributed technique used in [14] to compute the tree centers. Moreover, it is similar to the well known PIF algorithm for general topologies (see [18], and a comprehensive treatment in [19]). In a sense, `CIF` is dual to PIF: in the first phase of the PIF algorithm, a processor waits for the first message from any neighbor (instead of from $m-1$ neighbors), and then sends a message to all its other $m-1$ neighbors (instead of to the remaining neighbor). Similarly, in the second phase, a processor waits for a message from the latter $m-1$ neighbors, and then sends a message to the remaining (former) neighbor. Notice that the PIF algorithm works for any topology, and it is sufficient that one processor wakes up spontaneously. However, in PIF, if more than one processor wakes up spontaneously, the processors are not guaranteed to learn the value of the function on all inputs.

3 Consensus

In this section we prove that the complexity of `Consensus` is $2n$. This result applies for all cases considered in this paper: ids everywhere or at the terminals only, n of any parity known or unknown, and complexity in messages or bits.

THEOREM 3.1
For a tree with n links, BitC$(\texttt{Consensus}) \leq 2n$.

PROOF. Our $2n$ bits algorithm decides on the maximal bit over all input bits. Since this is an associative and commutative function of the input bit values, the `CIF` algorithm can be used, and the result follows from Theorem 2.1. ∎

The more difficult part is the matching lower bound. Actually, we prove a stronger claim: the lower bound holds for messages (and hence also for bits). We use the backward-preference scheduler described in Section 2.

THEOREM 3.2
For a chain with n links, MsgC(`Consensus`) $\geq 2n$. This bound holds when $M \geq 4$, if inputs are given to the terminals only, and when $M \geq 2n+2$, if inputs are given to all processors, and these bounds for M are tight.

PROOF. Consider an arbitrary concensus algorithm. Intuitively, our aim is to show that under the backwards-preference scheduler, there must exist an assignment of inputs to the terminals for which the waves generated by the algorithm meet, and then propagate back to the terminals. In fact, it is not hard to construct an algorithm that for certain inputs does send less than 2n messages. Our proof shows that there can be no more than just a few such inputs. We do this by a case analysis, considering the possible behaviors of the algorithm for particular inputs. The other lower bound proofs in this paper have a similar nature.

We say that a wave *covers* k links, if exactly k consecutive links, started at the wave originator, carried messages during the wave execution. Henceforth, we name the terminals S and T, and assume S wakes up spontaneously.

We begin with the setting where only terminals have id's. For any id N given to terminal S, let us consider the wave from S under the backward-preference scheduler, until the first moment when one of the following events happen:

1. the currently scheduled event is to deliver a message to T, and S has not decided yet (the wave reaches T before S decides),
2. the wave comes to a standstill before reaching T and before S decides, covering k links,
3. S decides before the wave initiated by it reaches T.

It is easy to see that these cases cover all the possibilities. We call the considered part of the execution *the wave caused by* N. Note that giving the same id N to T and waking it up leads to a symmetrical wave, because the program of T depends only on its input, all internal processors are identical, and the two directions from an internal processor are undistinguishable.

Consider case 1. Clearly, at least n messages were sent up to this moment, in order for the wave to propagate to T. By the backward-preference rules, there are no messages in transit (except for, may be, to T), and S has not decided yet. In order to decide, S must be activated, and since only T is active, at least n messages are needed for this. This results in at least $2n$ messages, as required. Therefore, if for at least one id, case 1 happens, the statement of Theorem is valid. Henceforth, we assume that there is no such id.

Assume next that there are at least two ids leading to waves in case 2. Give one of these ids to S and one to T. If the waves from S and T do not meet, we get to a global standstill situation where neither S nor T have decided. Therefore, there has to be a moment where the two waves meet, on a link e or at a processor. Consider the first case; the second is similar. At this moment S and T have not decided, and at least $n+1$ messages have been sent: two messages in opposite directions of e, and one message per link in each wave, allowing them to meet. So far S and T have not decided, and there are no messages in transit (except for over e). In order for S

and T to decide, they must be activated; for this, at least $n-1$ additional messages, propagating from e, are needed. Totally, this is at least $2n$ messages, as required.

Therefore, we can assume that there is no id for case 1, at most one id for case 2, and hence at least 3 ids for case 3. This implies (since the $M \geq 4$ ids are consecutive) that there is at least one id for each parity in case 3, and a total of at least two ids in each parity. Hence S decides necessarily on its own bit. Indeed, otherwise, giving to T any id with the same parity and considering the backward-preference execution from S, we arrive at an illegal decision of S.

Let us consider an execution where the id's at S and T are of different parities, and each of them generates a wave of type 3. Assume first that these waves do not meet. This is impossible because we have just argued that a terminal decides on its own bit if its wave has not been influenced by the other wave. It follows that for every such pair of ids of different parities, if any one is given to S and the other to T, and both waves start executing, there will be a moment when the waves meet on a link e (or at a processor, which can be proved in a similar way). By definition of case 3, neither S nor T has decided at this moment, and since we assume the backward-preference scheduler, the only messages in-transit are these two on e. Thus, we are in the same situation as above, where $n+1$ messages have been sent for the waves to meet, and in order for S and T to decide, messages must propagate from the end-points of e outwards to S and T, sending $n-1$ additional messages.

Consider now the case when inputs (ids) are given to all processors. We describe the changes in the proof only. Now, a wave from S is well defined if not only the id of S is fixed, but also ids of all other $n-1$ processors, except for T. We analyze the cases when ids 0, 1, 2, or 3 are given to S, as above, and prepare four sequences with $n-1$ ids each, to complete them, as follows. We take the sequence of the $n-1$ remaining even id's $(4, 6, \ldots, 2n)$ and its reversal $(2n, 2n-2, \ldots, 4)$ to complement the inputs of 0 and 2, and the two similar sequences of remaining odd id's $(5, 7, \ldots, 2n+1)$ and $(2n+1, 2n-1, \ldots, 5)$ to complement the inputs 1 and 3.

When two waves, from S and T, are combined, the direct and inverse orders of the same id set fit each other exactly. If the two waves have disjoint id sets for internal processors, we glue them in such a way that in the part of the chain covered by each wave, there are inputs corresponding to it.

In order to see that the bounds 4 and $2n+2$ are tight, we show two examples. First, if only terminals get ids and $M=3$, i.e., $Z=\{0,1,2\}$ (and this fact is known to the processors), let all processors decide on 0 in any case, without any communication. By the problem definition, this is legal, since never both ids given to the terminals are odd. Second, if all processors get ids and $M=2n+1$, let also all processors decide on 0 in any case, without any communication. This is legal by the same reason, since there are $n+1$ processors, in the chain, and only n distinct odd ids. ∎

Remark. By the above proof, the lower bound of Theorem 3.2 holds even if only terminals are ought to decide.

4 Leader: Message Complexity

Recall that in the `Leader` problem each processor must know whether it is the leader, and if not, which incident link is on the unique path from it to the leader. In the next section, we deal with message complexity; bit complexity is considered in the section after it.

4.1 Known Size of the Network

Assume n is known to each processor. We present an algorithm that elects as the leader one of the tree centers. Recall from Section 2.3 that there is a single center if n is even and may be two centers if n is odd.

THEOREM 4.1
In a tree with n links, if n is known then

$$\text{MsgC}(\texttt{Leader}) \leq \begin{cases} n & \text{if } n \text{ is even,} \\ n+1 & \text{if } n \text{ is odd.} \end{cases}$$

PROOF. We use a variant of the `CI` algorithm described in Section 2.1 to compute the centers of the tree (a similar technique for finding them is used in [14]).

First consider the case of n even, and hence, of one center. The center is found by using as f in the `CI` algorithm the number of processors in the tree. Each processor starts with the "input" value $v = 1$ in the `CI` algorithm. When an intermediate processor P receives a message from a neighbor Q with a value u, it knows that u is equal to the number of processors in the subtree hanging from it on (P, Q), rooted at Q. It computes $v \leftarrow v + u$, the number of processors in the subtree rooted at P including all processors from which it has received messages so far. When P has received messages from all but one of its neighbors, say R, it learns also the number of processors in the subtree hanging from it on (P, R), rooted at R, since it knows n: it is equal to $n + 1 - v$, (where v holds the current value of P's local variable). Thus P knows whether it is the center of the tree (when all the subtrees hanging from it have weight less than $n/2$). If so, it decides to be the leader. Otherwise, P decides that the leader is in the direction towards R, and it sends its value v to R. The number of messages sent is n: one message per link in the direction towards the center of the tree.

Now, in the case of n odd there may be two centers. In order to break the symmetry and elect one of them, the `CI` algorithm above is extended to compute in parallel the maximum id of a terminal. That is, each processor, in addition to the value v described above, computes another value, v', equal to the maximum id at a terminal in the corresponding subtree. Every message a processor sends contains both v and v'. When a processor receives a message with values u, u', it executes $v \leftarrow v + u$, and $v' \leftarrow \max(v', u')$. If there are two centers, each central processor recognizes that it is central, and waits for the message from the other central one. The central processor that has sent the maximal id to the other one decides that it is the leader. For any non-leader, the leader is in the direction to its parent. The total number of messages is $n + 1$: one message per link in the direction to the nearest center, and two messages on the link connecting the two centers.

Note that the above algorithm ignores ids of non-terminal processors, if any.

Remark. In the case when all processors have ids, it is possible to reduce the size of almost all messages as follows. Each message, except for the two messages on the link connecting the two centers, can contain the subtree weight only. On this exceptional link, if any, each one of the centers sends also its id; the center with the greater id wins. ∎

The following shows that the bound of the previous theorem is tight.

THEOREM 4.2
In a chain with n links, if n is known then

$$\text{MsgC}(\texttt{Leader}) \geq \begin{cases} n & \text{if } n \text{ is even,} \\ n+1 & \text{if } n \text{ is odd.} \end{cases}$$

This bound holds when $M \geq 4$, if inputs are given to the terminals only, and when $M \geq 2n+2$, if inputs are given to all processors.

PROOF. The intuition of the proof is similar to that of Theorem 3.2, but simpler: We show that under backward-preference executions, there must exist two ids in which the waves meet. We describe the proof for the case of odd n. The other case is similar.

We begin with the case when only terminals have ids. Assume that there are at least three ids causing the wave to get to a standstill *before* sending any message over the middle link. Then, for two of the ids, the same situation is reached: a leader from the processors participating in the wave generated by the id is or is not chosen. We give these two ids to S and T and execute the waves from both terminals. By the assumption, the waves do not reach each other before the global standstill. Moreover, by our choice, either two leaders are chosen, or no leader is chosen, a contradiction.

Therefore, we can assume now that there are at most two ids causing the wave to get to a standstill before sending any message over the middle link. Hence, there are at least two ids such that the wave from them causes sending a message over the middle link. We give the corresponding ids to S and T, and execute the waves so that the first messages over the middle link, one from each side of the chain, are sent simultaneously. Clearly, up to this moment at least $n+1$ messages are sent in the execution, as required.

The case when inputs (ids) are given to all processors is considered similarly to that in the proof of Theorem 3.2 (in fact, considerations now are a bit simpler). ∎

4.2 Unknown Size of the Network

This case also concerns message complexity, but now nothing is known on the value of n. In this case, it is not sufficient to execute just the collect phase of the algorithm. This implies a higher complexity. In fact, the complexity is high also in terms of bits: our algorithm sends an id in every message, and thus, its bit complexity is $2n \log M$. We know that the high cost $\Omega(n \log M)$ is unavoidable from the results of [7].

THEOREM 4.3
In a tree with n links, if n is unknown then MsgC$(\texttt{Leader}) = 2n$. The lower bound holds when $M \geq 3$, if inputs are given to the terminals only, and when $M \geq 3n$, if inputs are given to all processors.

PROOF. For the upper bound, our algorithm elects the terminal with the greatest id as a leader. Since maximum is an associative and commutative function of the input id values, the `CIF` algorithm can be used, by Theorem 2.1. Observe that when a processor receives the last message (from its parent), it learns not only the maximal id, but also the direction to it in the tree, as required.

For the lower bound in a chain where only the terminal processors have ids, we consider executions of the backward-preference scheduler where the terminal T is waken up spontaneously only in the case when the wave from S stops before reaching it. The proof is by contradiction, assuming that there are three ids such that the backward-preference execution from them ends in less than $2n$ messages.

For any such execution, there is at least one link on which messages are sent only in one direction, from S to T, during the entire execution. An important observation: All the processors between the terminal S, which starts the algorithm, and this link must decide before a message is delivered on this link, since they will not be activated after this moment. We glue two such waves propagating from the two chain terminals, by identifying two such links, and synchronize them so that the two waves meet on the identified link. Pay attention that thus we consider a chain of another length, in general; however, the assumption that n is not known implies that the glued execution is a legal execution of our algorithm. Notice that at the moment of sending the messages over the glued link, *all* processors have decided. Now, since there are 3 such ids, we can choose two waves such that either two leaders are elected, one among the processors of each wave, or no leader is elected. This gives the desired contradiction.

Assume now that ids are given to all processors. Then, for executing a wave as above, ids of at most n processors are used. We choose three disjoint sets of n ids each, for the three waves as above, which ensures using the same way of proof. ∎

5 Leader: Bit Complexity

In this section, we consider bit complexity bounds for `Leader` in the case of n even. This improves the results in [7], which presented a simple algorithm with the bit complexity $2n$. Here we show that the bit complexity is precisely $1.5n$. The case of odd n was treated in [7], where it was shown that the bit complexity is $\Theta(n \log M)$. This result relied on a $\Omega(n)$ lower bound which we prove here. Actually, we show that if n is odd then BitC$(\texttt{Leader}) \geq 1.5n$.

We start with the algorithms, which as in [7], do not use ids.

THEOREM 5.1
If n is known to be even then BitC$(\texttt{Leader}) \leq 1.5n$.

PROOF. We describe an algorithm which sends at most $1.5n$ bits for the tree topology using the `CI` strategy; it is similar to the one of Theorem 4.1, case n even. The function f to be computed is the parity of the number of processors in a subtree. Thus, each processor, P, starts with input $v = 1$, and computes modulo 2 the bits received plus its own bit; when there remains a single incident link without a bit received, P sends the computed bit over it to its parent in the execution. This leads to $n + 1$ bits sent: one bit per link, except for the last link, e_{last}, where two processors send one-bit messages to each other as to parents, in opposite directions. Since the bit sent from processor P over a link e to its parent Q is the parity of the number of processors in the subtree rooted at P hanging on e from Q, and since the number of nodes in the tree $n + 1$ is odd, the link e_{last} carries 0 in one direction and 1 in the other. The unique processor that sends 0 and receives 1 on e_{last} will be the leader.

Note that the above algorithm sends only $n + 1$ bits and elects a leader, but it does not terminate, except at the elected leader. We now modify it to send at most $1.5n$ bits, so as to let also each non-leader processor terminate and decide which of its incident links leads to the leader. For this, when a processor P sends a bit to its parent, Q, it acts as follows: First, it sends 0 to each of its children from which it received 0. In addition, if the bit sent by P to its parent Q is 1, P decides that it is not the leader and that the leader is in the direction of Q; then P terminates. If this bit is 0, then P waits until it receives a bit from Q. Upon receiving 0 from Q (the case when the "wave" propagates behind Q), P decides that it is not the leader and that the leader is in the direction of Q. Upon receiving 1 from Q (the case when $(P, Q) = e_{last}$), P decides that it is the leader. In both cases, P terminates.

Finally, we show that the total number of bits sent is at most $1.5n$. There are exactly $n + 1$ bit-messages which are sent by a processor to its parent: one on each link except e_{last}, on which two such messages are sent. Observe that each of the remaining messages, say from Q to P, is sent after P has sent 0 on the same link to its parent Q, indicating that there is an even number of nodes in the subtree rooted at P hanging on (Q, P) from Q; we call such a latter message a *forward 0-message*. The bit 0 sent on e_{last} has the same nature; we call also it a forward 0-message. Now, the number of forward 0-messages coincides with the number of links carrying two messages. Hence, for our purpose, it suffices to show that at most half of the links carry forward 0-messages.

To this end, consider a forward 0-message sent from P to Q. The sum of sizes of the subtrees rooted at the children of P must be odd, and thus at least one of them is odd. On the edge connecting each such child to P, no forward 0-message is sent. By mapping each link (P, Q) as above to one such latter link, we get a 1-1 mapping that pairs each edge which carries a forward 0-message to one that does not. Therefere, at most half of the edges carry forward 0-messages. This completes the proof.

We note that in the case of a chain, the numbers of bits sent over links are alternating 1 and 2, from any chain end to the wave meeting link; hence, in this case, *exactly* $1.5n$ bits are sent. ∎

THEOREM 5.2
In a chain with n links, if n is known to be even (but otherwise unknown), then BitC($\texttt{Leader}$) $\geq 1.5n$. This bound holds when $M \geq 5$, if inputs are given to the terminals only, and when $M \geq 5n$, if inputs are given to all processors.

PROOF. The proof is by a refinement of the proof technique of Theorem 4.3, which the reader is advised to recall. As in that proof, we show that if too few messages are sent, then we can glue two backwards-preference executions by a link on which a message is sent only in one direction, and get an illegal execution. To guarantee that such a glueing provides an even length chain, we show that there are executions in which *two consecutive links* carry only one message each.

Suppose, to the contrary, that there is an algorithm with better complexity. That is, we can choose some even length n_0 such that the algorithm is supposed to use strictly less than $1.5n_0$ messages in any execution in any chain of length n_0.

Assume first, that only terminals have ids. We use the backward-preference scheduler as in the proof of Theorem 4.3. Let us consider backward-preference executions from all ids in the chain of length n_0. Assuming there are at least 5 ids, one of cases 1 and 2 below must hold:

Case 1: There are three ids such that the wave from them stops before waking up the second terminal. Let us consider a $2n_0$ length chain with two such ids given to the terminals. The execution under the backward-preference scheduler with waves from both sides stops without wave meeting. Now, there are either two ids such that upon the termination a leader is elected in the area covered by this wave, or two ids for which no such leader is elected. Choosing such a pair, we get, in the first case, that two leaders are elected, and in the second one, that no leader is elected, a contradiction.

Case 2: If case 1 does not hold, there are three ids such that in the backward-preference execution from them, each link carries at least one message. Let us extend arbitrarily each such backwards-preference execution E to a full execution $\tilde{E}$, in which all processors decide. Let S denote the terminal-originator of these executions. By the assumption that less than $1.5n_0$ messages are sent in the entire $\tilde{E}$, there are two *consecutive* links, (P, Q) and (Q, R), with exactly one message sent on each, in $\tilde{E}$. Observe that the single message sent from Q to R is within E, by the assumption of Case 2; let m denote the moment of delivering this message. By definition of the backwards-preference execution, there is no message in-transit between S and R, at moment m. Also, no messages are sent from R to Q in the entire $\tilde{E}$, by choice of (Q, R). Therefore, after moment m, no processor between S and Q is activated in the entire $\tilde{E}$; hence, all of them have decided before moment m within E.

Let us consider, for any backward-preference execution chosen at Case 2, its initial part up to the moment as above; all processors participating in it, except for the last one, have decided. We can glue any two such execution parts by identifying two links, one in each link pair as above, so that the length of the resulting chain is even (we can do so even if we fix one of the links in one of these pairs, since we can select any of the two consecutive links in the other one). Also, we synchronize the glued executions so that the two messages on the identified link are sent simultaneously. It is easy to see that the glued execution in the new chain is legal. As observed above, at the moment of sending the messages over the glued link, *all* processors in the new chain have decided. Now, choosing two waves where a leader was elected inside the covered area or two waves where no leader was elected inside it, we arrive at a contradiction.

If inputs (ids) are given to all processors, the following change to the proof suffices. Observe that the parts of executions participating in our analysis are defined by at most n ids. We choose five disjoint sets of n ids each, for the five executions as above, which ensures using the same proof. ∎

Note that the above proof actually implies that, under the assumption of the Theorem, at least $1.5n$ *messages* must be sent.

The same proof strategy can be used to prove the following result for odd n. We note however that in [7], a stronger result is given, which shows that the bit complexity in this case is $\Theta(n \log M)$.

THEOREM 5.3
In a chain with n links, if n is known to be odd (but otherwise unkonwn), then BitC($\texttt{Leader}$) $\geq 1.5n$. This bound holds when $M \geq 5$, if inputs are given to the terminals only, and when $M \geq 5n$, if inputs are given to all processors.

6 Discussion

1. Our results show that when nothing is known about n, `Leader` is at least as hard as (and sometimes harder than) `Consensus`, both in terms of messages and bits; `Leader` is easier than `Consensus` when certain information about n, which can be used for symmetry breaking, is known.

2. The decision of distinguishing terminal processors both as possessing ids and as being able to wake-up spontaneously, seems natural, from the practical, as well as from the theoretical point of view. Suppose, for example, that the terminals are important processors (e.g., users), while the other ones are interconnection stations that only relay messages (see also [6] for such a point of view). Also in communication complexity studies [5, 20], Alice and Bob, the chain terminals, are the only processors with inputs.

Further work is needed to investigate other alternatives; the assumption of which processors are subject to waking up spontaneously can affect the complexity of a task. For example, a natural assumption could be: at least one processor wakes up spontaneously. Clearly, our lower bounds hold under this assumption, but our algorithms do not.

3. If n is known to be even, we have shown that $1.5n$ bits are necessary and sufficient to solve `Leader` in a chain. It is an open question whether this bound holds if n is even and its actual value is known.

4. All our lower bounds for the tree topology are proved for a chain. It may be of interest whether these bounds can be proved for other special tree topologies.

5. It may be interesting to check tightness of bounds on M used in our lower bound Theorems for `Leader`.

References

[1] Hagit Attiya and Jennifer Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics,* McGraw–Hill, England, 1998.

[2] Hans L. Bodlaender, "New lower bound techniques for distributed leader finding and other problems on rings of processors," *Theor. Comput. Sci.* **81** (1991), 237–256.

[3] Hans L. Bodlaender, Shlomo Moran, Manfred K. Warmuth, "The distributed bit complexity of the ring: From the anonymous to the non-anonymous case", *Information and Computation* **114** (2), pp. 34–50, 1994.

[4] James E. Burns, "A formal model for message passing systems," Technical Report TR-91, Computer Science Dept., Indiana University, Bloomington, September 1980.

[5] Martin Dietzfelbinger, "The linear-array problem in communication complexity resolved," in *Proceedings of the 29th ACM Symposium on Theory of Computing*, pp. 373–382, 1997.

[6] Ye. Dinitz, T. Eilam, Sh. Moran, and Sh. Zaks. "On the Total_k-Diameter of Connection Networks," *Theoretical Computer Science* **247** (2000), no. 1–2, 213-228.

[7] Yefim Dinitz, Shlomo Moran, Sergio Rajsbaum, "Bit Complexity of Breaking and Achieving Symmetry in Chains and Rings," in *Proc. of the 31st ACM Symp. on the Theory of Computation,* pages 265–274, 1999.

[8] Yefim Dinitz, Shlomo Moran, Sergio Rajsbaum, "Exact Communication Costs for Consensus and Leader in a Tree," 7th International Colloquium on Structural Information and Communication Complexity (SIROCCO), L'Aquila, Italy, June 20-22, 2000; Proc. in Informatics 7, M. Flammini, E. Nardelli, and P. Spirakis (Eds.), Carleton Scientific Press, pp. 63–77.

[9] M. Fischer, N. Lynch, M. Paterson, "Impossibility of distributed concensus with one faulty process," *J. Assoc. Comput. Mach.,* **32**(2):374–382, Apr. 1985.

[10] P. Flocchini, B. Mans, N. Santoro, "Sense of direction: formal definition and properties," In Proc. of 1st Colloquium on Structural Information and Communication Complexity (SIROCCO), Ottawa, Canada, 1994, pp. 9–34.

[11] Robert G. Gallager, Pier A. Humblet, Philip M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Prog. Lang. Syst.,* **5**(1):66–77, Jan. 1983.

[12] Daniel S. Hirschberg, J.B. Sinclair, "Decentralized extrema-finding in circular configurations of processes," *Commun. ACM* **23** (11), pp. 627–628, 1980.

[13] Ephraim Korach, Shlomo Moran, Shmuel Zaks, "Tight upper and lower bounds for some distributed algorithms for a complete network of processors," *Proc. of the 3th Annual ACM Symp. Princ. Dist. Comp.* (1984), pp. 199–207.

[14] Ephraim Korach, D. Rotem and N. Santoro, "Distributed algorithms for finding centers and medians in networks," *ACM Trans. Program. Lang. Syst.*, **6** (3), pp. 380–401, 1984.

[15] Eyal Kushilevitz, Nathan Lineal, Rafail Ostrovsky, "The linear-array conjecture in communication complexity is false," *Proc. of the 28th Annual ACM Symp. Th. Comp.* (1996), pp. 1–10.

[16] Eyal Kushilevitz, Noam Nisan, *Communication Complexity*, Cambridge University Press, 1997.

[17] Nancy A. Lynch, *Distributed Algorithms,* Morgan Kaufmann Publishers, Inc. 1996.

[18] Adrian Segall, "Distributed network protocols," *IEEE Trans. on Information Theory,* IT-29, 2 (Jan. 1983), pp. 23–35.

[19] Gerard Tell, *Distributed Algorithms,* Cambridge University Press, 1994.

[20] Prasoon Tiwari, "Lower bounds on communication complexity in distributed computer networks," *J. Assoc. Comput. Mach.* **34** (1987), pp. 921–938.

[21] Andrew C. Yao, "Some complexity questions related to distributed computing," *Proc. of the 11th ACM Symp. on Theory of Comp.* (1979), pp. 209–213.

Received submission date