Bit Complexity of Breaking and Achieving Symmetry in Chains and Rings *

Yefim Dinitz[†] Shlomo Moran[‡] Sergio Rajsbaum[§]

December 10, 2007

Abstract

We consider a failure-free, asynchronous message passing network with n links, where the processors are arranged on a ring or a chain. The processors are identically programmed but have distinct identities, taken from $\{0, 1, \ldots, M-1\}$. We investigate the communication costs of three well studied tasks: `Consensus`, `Leader`, and `MaxF` (finding the maximum identity). We show that in chain and ring topologies, the message complexities of all three tasks are the same. Hence, we study a finer measure of complexity: the number of transmitted *bits* required to solve a task T, denoted $BitC(T)$.

We prove several new lower bounds (and some simple upper bounds) that imply the following results: For the two processors case, $BitC(\texttt{Consensus}) = 2$ and $BitC(\texttt{Leader}) = BitC(\texttt{MaxF}) = 2\log_2 M \pm O(1)$, where the gap between the lower and upper bounds is almost always 1. For a chain, $BitC(\texttt{Consensus}) = \Theta(n)$, $BitC(\texttt{Leader}) = \Theta(n + \log M)$, and $BitC(\texttt{MaxF}) = \Theta(n \log M)$. For the ring topology, we prove the lower bound of $\Omega(n \log M)$ for `Leader`, and (hence) `MaxF`.

We consider also a chain where the intermediate processors have no identities. We prove that $BitC(\texttt{Leader}) = \Theta(n \log M)$, which is equal to n times the bit complexity of the problem for two processors. For the specific case when the chain length is even, we prove that $BitC(\texttt{Leader}) = \Theta(n)$, for both above settings. In addition, we show that for any algorithm solving `MaxF`, there exists an input, for which *every* execution has the bit complexity $\Omega(n \log M)$ (this is not the case for `Leader`).

In our proofs, we use both methods of distributed computing and of communication complexity theory, establishing new links between the two areas.

---

*A preliminary version of this paper appeared in [7].

†Department of Computer Science, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105 Israel. Part of the work was done while this author was on leave from the Technion, Haifa, Israel at Instituto de Matemáticas, UNAM, México. Email: `dinitz@cs.bgu.ac.il`.

‡Department of Computer Science, The Technion, Haifa, 32000 Israel. This work has been supported in part by the Bernard Elkin Chair for Computer Science. Part of the work was done while this author was at the University of Arizona, supported by US-Israel BSF grant 95-00238, while he was visiting the Instituto de Matemáticas, UNAM, México, and while he was visiting the Institute of Information Sciences, Academia Sinica, at Taipei. Email: `moran@cs.technion.ac.il`.

§Instituto de Matemáticas, UNAM, Ciudad Universitaria, D.F. 04510, México. Partially supported by DGAPA-UNAM Projects. Email: `rajsbaum@math.unam.mx`.

1 Introduction

1.1 Motivation from Communication Complexity

The basic problem in the area of *communication complexity* was introduced by Yao in 1979 [22]. This problem asks what is the number of bits that two processors, A and B, have to communicate to each other in order to compute a function $f(x, y)$ of their respective private inputs: x and y. The model assumes that they send bits one at a time, starting with some bits sent by A, then some bits sent by B, and so on. At the end of the computation, both processors know the value of $f(x, y)$. For example, if $f(x, y) = \max(x, y)$ and x, y are integers in $Z_M = \{0, 1, \cdots, M - 1\}$, then it is known that the *communication complexity* of f is exactly $\lceil \log M \rceil$; i.e., this number of bits must be transmitted to solve the problem in the worst case. There exist generalizations of this problem to networks containing more than two processors. Traditionally, in this model, it is assumed that each of the processors in the network has a complete knowledge of the topology of the network and of the identities of all other processors. This justifies the assumption, for the basic problem, that a certain processor starts the computation, and that at any moment there is a transmission in only one direction of the link. Many results exist on Yao's basic problem and variants of it (e.g., [21, 14, 5]). An introduction to the area can be found in [16].

Consider now a seemingly slight modification of the problem of computing $\max(x, y)$, which belongs to the *distributed computing* area ([1, 17, 20]). This time, the two parties are two identical (unnamed) processors, each one provided with a value in Z_M—its *identity* (*id*); the two processors have to find the larger identity. Note that in this scenario one cannot fix, ahead of time, a processor that will send the first bit in the communication. This makes the problem more complex. Indeed, our results imply that at least $2\lfloor \log M \rfloor - 3$ bits must be transmitted to solve the problem in this setting. This introduces a new parameter to the problem, namely, the information on the network known to processors. In more general networks, we may also assume that processors do not know the structure of the entire network and/or the number of processors in the network. These are the usual assumptions in the distributed computing literature. Moreover, in this literature, it is common that the problem which has to be solved is not necessarily a function of the processor inputs, but is specified by a *task:* an input-output relation (several outputs are allowed for the same input, see [19]), where the output is not required to be the same for all processors. The inputs usually include the processor identities which are distinct and are not mutually known. When considering tasks, the difference in power between the two models can become quite dramatic: the `Leader` problem is solvable in the communication complexity model with 0 bits (A outputs 1, B outputs 0), while in the distributed computing model it requires at least $2\lfloor \log M \rfloor - 3$ bits.

1.2 The problem and related work

The two most studied tasks in the distributed computing literature are probably `Consensus` and `Leader`. A large number of results on these

tasks exist, including algorithms, lower bounds and applications, in a variety of distributed computing models. However, almost always, `Consensus` is studied in the shared memory setting, mainly from the fault tolerance point of view, while `Leader` is studied in the message passing setting, emphasizing message complexity bounds. We view these tasks as duals in the sense that in `Consensus` all processors have to agree on the same value, while in `Leader` one processor has to output a different value from the rest of processors. In this sense, `Consensus` is about achieving symmetry, while `Leader` is about breaking symmetry. In this paper we study the communication cost of these two tasks and of the variant of `Leader`, called `MaxF`, in which the elected processor must have the largest id in the system.

In the distributed computing area, the usual communication cost measure is message complexity. Under this measure, `Leader` has been extensively studied in an asynchronous, failure-free, message passing distributed system. It is known that, in general, $\Theta(m + n \log n)$ messages are necessary and sufficient for this task in a network with n processors and m communication links (e.g., see [12] for an upper bound and [4] for a lower bound), and this bound holds even when it is given that the processors are arranged in a ring. Some other special topologies have lower message complexities; e.g., trivially, $\Theta(n)$ is the complexity for trees, and $\Theta(n \log n)$ is the complexity in complete graphs [15]. In all these cases, `Leader` and `MaxF` have the same message complexity.

The message complexity of `Consensus` was not studied prior to the conference version of this work [7]; it was studied by other authors later (e.g. the research line of [11]), see discussion in the next paragraph. It is easy to see that `Consensus` is not harder than `Leader`: in any topology, once a leader has been elected it can broadcast the value to be decided. Thus, only additional $O(m)$ messages are needed to solve `Consensus` (in some cases, like in a complete graph topology, only $O(n)$ additional messages are needed). We start by observing that the opposite is not true: The message complexity (up to a constant factor) of `Leader` is not reduced even if `Consensus` is given for free. In view of this, it may be surprising that, as we show, the message complexity of `Consensus` is the same as that of `Leader`. This motivates the use of the following finer measure to distinguish between the communication costs of these tasks: the number of bits sent, which we call *bit complexity.* With respect to this measure, roughly speaking (below we describe more precisely the results), we show that in chains, `Consensus` is easier than `Leader`, which is easier than `MaxF`.

The bit complexity of tasks like `Consensus` and `Leader` in our distributed setting was not studied before. It was studied in some related settings. The bit complexity measure of functions in a distributed setting was studied earlier in [18, 3], where it was shown that computing any non-trivial function on a ring of n processors requires $\Omega(n \log n)$ bits. The bit complexity of computing functions by two distinct and mutually known players was studied extensively in the last few decades (see e.g. [22, 21, 16]); some of the techniques developed there are applied in this paper. Besides, there is a line of research devoted to finding efficient `Consensus` algorithms, in particular with low bit complexity. Mostly synchronous models have been considered in this research, and a complete graph topology where processors can fail is assumed (see

[11] for a recent publication in this line, and references herein).

By the reduction from `Consensus` to `Leader` mentioned above, in chains and rings, a solution to `Leader` of complexity $f(n)$ implies a solution to `Consensus` of complexity $O(f(n) + n)$, for both message and bit complexity. This fact implies two challenges:

- for cases when the complexity of `Leader` is more than linear, to show that the complexity of `Consensus` is strictly less than that of `Leader`, and
- for cases when the complexity of `Leader` is linear, to clear up the correct order of the complexities of `Consensus` and `Leader`.

This paper is concerned mostly with achieving the first goal, and deals with asymptotic ("order of") complexity measures. In the paper [8], we concentrate on the second goal, for the tree (in particular, chain) topology; there, exact complexity bounds are achieved. It is worth to mention that among the settings considered in [8], all the three possible cases: the complexity of `Consensus` exceeds that of `Leader`, the inverse case, and the case of equal complexities, are encountered.

1.3 The results

The model we consider consists of a failure-free, asynchronous message passing distributed system, with arbitrary but finite link delays and negligible local computation times. We assume that processors have distinct ids, and that processors are identical in the sense that they all run the same program. However, the processors may behave differently, because the program of a processor has access to its id and its number of incident links. The topologies that we study are: a chain, with n links and $n+1$ processors, and a ring, with n links and n processors. The ids are taken from the set $Z_M = \{0, 1, \ldots, M-1\}$. For a task T, $BitC(T)$ is the number of bits needed to solve T in the worst case.

1.3.1 Message complexity

In Section 3, we prove two $\Omega(n \log n)$ lower bounds on the message complexity of `Consensus` in a ring; one for the case where n is unknown to the processors, and a more complicated one for the case where n is known to them. This lower bound implies that all three tasks have the same message complexity on a ring, $\Theta(n \log n)$. This is because there exists an $O(n \log n)$ algorithm for `MaxF`, see [13]; thus, the same upper bound holds for the other two problems. The matching lower bounds for `Leader` and `MaxF` are given in [4] (for the case where n is unknown) and in [2, 10] (for the case where n is known). We stress that the lower bounds for `Leader` and `MaxF` hold even if `Consensus` is given for free (see Section 2).

For the chain topology, it is easy to see that all three tasks have $\Theta(n)$ message complexity.

1.3.2 Bit complexity

The rest of the results are on bit complexity. The proofs for `Consensus` are simple. Most of the paper is devoted to `Leader` and `MaxF`.

Achieving Symmetry in a Chain. Simple arguments show that in a chain $BitC(\texttt{Consensus}) = \Theta(n)$. See Section 4.1 for the case of $n = 2$, and Section 5 for the general case.

Breaking Symmetry for Two Processors. In Section 4.2, we show that `MaxF` (and thus `Leader`) can be solved with $2\lceil \log M \rceil - 2$ bits, and every `Leader` algorithm (and thus every `MaxF` algorithm) requires at least $2\lfloor \log M \rfloor - 3$ bits.[1] Notice that this is twice larger than the communication complexity of any function $f(x, y)$, $x, y \in \{1, \cdots, M\}$ (Proposition 1.3 in [16]); i.e., when the processor names are mutually known.

Breaking Symmetry in a Chain. These results are in Section 5. First, we show that when the chain is of even length, then $BitC(\texttt{Leader}) = \Theta(n)$.

The case of odd length is more interesting. We present an $O(n + \log M)$ bits algorithm for the general case. By proving a matching $\Omega(n + \log M)$ lower bound for `Leader` in the odd length case, we show that in this case (and thus, in the general case) `Leader` is harder than `Consensus`.

Breaking Symmetry in a Chain: Two-Inputs Case. The most interesting setting for `Leader` is when ids are given only to the two terminals of a chain, while the remaining processors have no inputs (and thus are completely identical). The same question has been considered in the communication complexity literature [21, 14, 5]. It was proved in [5] that the communication complexity of a function $f(x, y)$ of two inputs located at the end processors of a chain of length n equals n times the communication complexity of $f(x, y)$ on a chain of two processors (up to a multiplicative factor). Our main theorem is in Section 6, where we show that an analogue of this result holds for `Leader` in a chain of odd length, i.e., $BitC(\texttt{Leader}) = \Theta(n \log M)$. Such a result does not hold in a chain of even length, since the $\Theta(n)$ algorithm mentioned above works also in this two-inputs setting.

Breaking Symmetry in a Chain: the `MaxF` task. A trivial algorithm shows that $BitC(\texttt{MaxF}) = O(n \log M)$. It follows from [21] that the communication complexity of finding the largest input in a chain is $n \log M - O(1)$. Hence, $BitC(\texttt{MaxF}) = \Theta(n \log M)$ also for the distributed computing two-inputs model. We show in Section 6.3 that for the two-inputs case, $BitC(\texttt{MaxF}) = \Omega(n \log M)$ even for the variant of the complexity definition, where the *best possible*, instead of the worst possible execution, for any algorithm on any input, is considered. The proof of this statement works, with minor changes, also if all processors are given inputs (this observation is joint with Noam Solomon). As a consequence, also for this regular distributed computing setting, $BitC(\texttt{MaxF})$ is $\Omega(n \log M)$. Therefore, `MaxF` is harder than `Leader`, in a chain.

Rings. For the ring topology, the best known upper bound for `Leader`, `MaxF` and `Consensus` is implied by the algorithm of [13], which has bit complexity $O(n \log n \log M)$. As for lower bounds, all these three tasks have the $\Omega(n \log n)$ bits lower bound implied by the bound $\Omega(n \log n)$ on their message complexity [4, 2, 10]. For `Leader`, and hence also for `MaxF`, we (slightly) improve this lower bound to $\Omega(n \log M)$, in Sec-

[1] The first author and N. Solomon [9] closed the gap between these bounds, by suggesting an optimal algorithm, for this problem, and proving its optimality.

tion 7.

1.4 About our techniques

Here we describe the three lower bound techniques of this paper. The second and third ones relate for the first time two areas: distributed computing and communication complexity.

Our first technique is used for proving lower bounds on breaking symmetry in a two-processor chain and in a ring, in Sections 4.2 and 7, respectively. It consists of finding long symmetric synchronous executions. A lower bound is obtained because no symmetric execution can break symmetry.

Our second technique is presented in Section 6. It is developed for proving a Cut-and-Paste property, analogous to that used for achieving the lower bound in [5]. In contrast to the model of [5], in our case we need to choose a specific scheduler, which produces a unique execution, for any algorithm and any given input. We then consider executions under this scheduler only, since to prove a lower bound it is sufficient to prove it over a subset of all executions. However, fixing a scheduler induces a delicate problem when proving the Cut-and-Paste property (Lemma 6.1): it must be shown that the "pasted" execution is indeed the execution under this particular scheduler.

Our third technique is a modification of the partition-into-rectangles technique of [22, 21, 5] (see also [16]) for a chain of two processors; it is described in Section 4.3. We need to adapt this technique to our case, where processors always have distinct inputs, since the technique as applied in communication complexity assumes that processors can get all combinations of inputs from a set of values. Also in contrast to our model, in communication complexity the algorithm computes a function, and all executions of the algorithm on a given input produce the same history on each communication link. To establish these two properties in our model, we consider executions of a leader election algorithm under a fixed, specific scheduler, and show that these executions compute some anti-symmetric function. Then, we show that such a function implies a partition of the matrix into a large number of so called "semi-rectangles."

2 Model Definition and Preliminaries

Many notions presented in this section are standard and appear in more detail in textbooks such as [1, 17, 20].

We assume the usual asynchronous, failure-free message passing distributed model consisting of a network of processors, some of which are pairwise connected by links. We consider two network topologies in this paper: a chain and a ring. Message transmission delays are arbitrary but finite. Also, links are FIFO; that is, for any link and any direction on it, messages are delivered to a processor in the order in which they were sent. The number of links is denoted by n, so that in the ring there are n and in the chain $n+1$ processors. The processors have distinct identities (ids) from the set $Z_M = \{0, 1, \ldots, M-1\}$, so M must be at least n or $n+1$ (although in Section 6 we consider a

chain with ids only at the terminals).

A *distributed algorithm* consists of a set of identical sequential, deterministic algorithms, one for each processor. The algorithms of the processors are identical in the sense that they depend only on the processor input and number of incident links. In this paper, the processor input consists of its id only, if any.[2] The *configuration* of the system at a given time specifies the local states of the processors and the messages in transit at the links. In the *initial configuration* all processors are *asleep* and in the same initial state, except for the id and number of incident links information. When a processor wakes up, spontaneously, or when it receives a message on an incident link, it is *activated*. Then, according to its algorithm and local information, it decides what messages to send on its incident links (it may send zero or more messages on each link), and changes its state. Messages are sent by a processor only when it wakes up or as an immediate response to receiving a message; that is, internal computation time is negligible.

A *scheduler* is a formal device that specifies the order in which processors wake up and messages are delivered. An *execution* of an algorithm is defined by a given scheduler in the obvious way: the next configuration is determined by the current configuration, the set of sleeping processors waken up (if any), and the messages in-transit delivered to their destinations (if any). In this paper, we consider only deterministic schedulers, hence, any scheduler produces a unique execution for each given initial configuration. In a chain topology, we assume that the terminals eventually wake up spontaneously, if not activated previously by a message.

We consider the following tasks:

1. `Consensus`: all processors must output the same bit. If all the ids are odd they must output 1, if all ids are even they must output 0.
2. `Leader`: one processor outputs 1, the rest output 0.
3. `MaxF`: the same as `Leader`, except that the processor to output 1 must be the one with the maximal id.

It is said that a distributed algorithm *solves a task* if for *any* initial configuration, where processors get distinct ids from Z_M, and at *any* execution (i.e., for any scheduler), every processor eventually outputs a bit, and these outputs satisfy the task specification.

The hardness of a task is evaluated according to the following complexity measures:

- *Message Complexity*: the number of messages needed to send, in order to solve the task, and
- *Bit Complexity*: the number of bits needed to send, in order to solve the task.

For a fixed *execution* solving the task, the meaning of such a complexity is obvious. For a fixed *algorithm* solving the task, in this paper, we study its *worst case complexity*. That is, we refer to the worst case,

[2] In some models, like the one used in communication complexity, it is assumed that the identities(=names) are fixed and mutually known, while each processor has a private input, in addition, which is known only to itself.

over all possible inputs and all possible schedulers. Hence, in order to obtain, for a given algorithm, some lower bound L, it is sufficient to fix a certain scheduler S and to prove existence of some input I, s.t. the algorithm sends at least L messages/bits, totally, upon I and under S.

The *complexity of a task* is defined as the best algorithm complexity, over all algorithms solving it. Accordingly, the complexity of any algorithm solving some task is an *upper bound* for the complexity of that task. On the other hand, in order to prove a *lower bound* for the complexity of some task, we should prove that lower bound for all algorithms solving it.

Let us introduce now some additional observations and assumptions, which we use in this paper.

1. In Section 6.3, we consider the *best scheduler* complexity, defined as follows. The greatest number of bits sent by an algorithm, over all inputs, is considered for every scheduler, and this measure is *minimized* over all schedulers. Proving a high lower bound for the best scheduler complexity shows that even a free choice of a scheduler does not help to reduce the number of bits sent.

2. Notice that we do not use the concept of time, in defining distributed computing; instead, our systems are event driven. Sometimes, in our analysis, we introduce a formal clock, for constructing a specific scheduler, as follows: the initial configuration corresponds to time 0, while passing to each next configuration increments the time by one. Also, sometimes we give names to processors, for convenience of algorithm analysis. However, those clock and names are not available to processors, and are in no sense related to algorithms.

3. In some of our proofs, we extend an algorithm so that the last bit sent by a processor is its output value. Such an algorithm is called *reporting*. Any algorithm can be extended in this way, at the cost of the number of extra bits required to send.

4. When we want to compute an exact bound on the bit complexity of a task, we assume that any message is a single bit. Thus, though a processor can send more than one bit as a response to a message, those bits are not guaranteed to be received at the same time. This avoids the situation where the same sequence of bits can be sent and received in more than one way, by segmenting it into different message sequences (note that there are 2^{k-1} ways to split a k bits sequence into consecutive messages).

Let us see that the assumption that any message is a single bit is not restricting, when bounds up to a constant factor are looked for. Consider the general setting, where messages have different lengths. First, we can introduce messages consisting of a single special symbol. This can be implemented by the price of just a constant factor to the total length, e.g., as follows: to encode 0 as 00 (i.e., two consequent messages with 0), 1 as 01, and the special symbol by 1. In this way, the general case can be reduced to the single bit per message setting, by encoding each end-of-message by this special symbol. As a result, there is no loss of information, in comparison to the general case, while the price is a constant factor to the total length only.

Finally, we give an evidence on the natural hardness order of `Consensus` as compared with `Leader` and `MaxF`. Let us show a simple reduction of `Leader` and `MaxF` to a set of inputs in which a (legal) decision of

`Consensus` is given for free. For any instance of the original problem $\mathcal{P}$, that is for any assignment of ids to processors, let us form the new ids by extending the old ones by bit 0 at the right. These ids, together with the decision 0 for `Consensus`, form an instance for the second version of the problem, $\mathcal{P}'$. Obviously, any algorithm $\mathcal{A}'$ for the second version run on such instances induces a general algorithm $\mathcal{A}$ for the original problem $\mathcal{P}$, of the same complexity. Moreover, the complexity $C(\mathcal{A}, n, M) \leq C(\mathcal{A}', n, 2M)$. Therefore, the complexity of $\mathcal{P}$ is bounded as

$$C(\mathcal{P}', n, M) \leq C(\mathcal{P}, n, M) \leq C(\mathcal{P}', n, 2M) \ .$$

That is, the complexities of $\mathcal{P}$ and $\mathcal{P}'$ are the same, if polynomial and if considered up to a constant factor. In our considerations, complexities are polynomial in n and $\log M$. Hence, the asymptotic complexities of $\mathcal{P}$ and $\mathcal{P}'$ are equal, even when multiplicative factors are taken into account.

3 Message complexity of achieving symmetry

In this section we present a lower bound on the number of messages needed to solve `Consensus` in a ring. This is the only section where we study message complexity; in the rest of the paper we deal with bit complexity only.

Any distributed algorithm for `Leader` in a ring can be modified to solve `Consensus` using $O(n)$ additional messages: first elect a leader, and then the leader sends a message around the ring informing all processors that the consensus decision value is the parity of its id. Since there is an $O(n \log n)$ messages algorithm for `Leader` [13] (where processors do not need to know n), there is an $O(n \log n)$ messages algorithm for `Consensus`, in a ring. We present two proofs for a matching lower bound: one for the case where n is unknown to the processors, and a more involved one for the case where n is known to the processors. Therefore holds:

Theorem 3.1 *The message complexity of* `Consensus` *in a ring is* $\Theta(n \log n)$.

3.1 Lower bound: unknown ring size

Theorem 3.2 *The message complexity of* `Consensus` *in a ring where* n *is unknown to the processors is* $\Omega(n \log n)$.

Proof: Consider any `Consensus` algorithm for a ring. An execution is called *open* at an edge e if no message has been delivered on e, and if all processors are quiescent (either waiting for a message or finished the algorithm). A 0-*ring* (resp., 1-*ring*) is one where all processors have even (resp., odd) ids.

Lemma 3.3 *If there exists an open execution on* i*-ring, where some processor has decided, then in every open execution on* $(1-i)$*-ring, no processor has decided, for* $i = 0, 1$.

Proof: Assume for contradiction that for $i = 0, 1$, there are i-rings C_i with open execution α_i at e_i, such that in both rings some processor has decided. Since no message has been delivered in e_0 or e_1, and the ids in the two rings are different, we can paste the two rings into a bigger ring C, and consider the execution $\alpha = \alpha_0, \alpha_1$ consisting of both open executions. In any infinite extension of α, all processors have to decide, but since the decision values are irrevocable, at least one of the processors that belonged to C_0 decides 0 and at least one that belonged to C_1 decides 1, a contradiction. ∎

Assume without loss of generality, that in every 0-ring, in every open execution no processor has decided. The proof of Theorem 3.2 is completed using the idea of Burns [4]. We sketch it here for completeness. We prove by induction on n that every 0-ring has an open execution where no processor has decided and $f(n)$ messages have been sent, where f is a nondecreasing function satisfying $f(n) \geq 2f(\lfloor n/2 \rfloor) + n/2$. Thus $f(n) = \Omega(n \log n)$. Consider a 0-ring of size n, and two disjoint segments of it, of sizes $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$. By induction hypothesis, there are open executions on each segment, which send at least $f(\lfloor n/2 \rfloor)$ messages. Pasting together the two executions, we obtain an execution, where at least $2f(\lfloor n/2 \rfloor)$ messages are sent and all processors are quiescent. If we extend this execution, by delivering messages along the open edges, eventually all processors have to receive a message and decide. Therefore, for at least one of the open edges, when we start delivering messages along it (while the other edge remains open), the two waves to the two directions from it (one of them may be empty) spread over at least $n/2$ of the processors. We extend this execution until all processors are quiescent, without delivering any message on the other edge. By Lemma 3.3, no processor can decide. In this way, we can obtain an open execution, where at least $2f(\lfloor n/2 \rfloor) + n/2$ messages are sent. ∎

3.2 Lower bound: known ring size

In this subsection, we improve the message complexity lower bound to the case where n is known:

Theorem 3.4 *For a ring of n processors that know n, the message complexity of* `Consensus` *is $\Omega(n \log n)$.*

The proof is based on the technique in [2]. Consider any `Consensus` algorithm $\mathcal{A}$ for a ring of size n. Let I be a set of identities. We say that $\mathcal{A}$ is *active on I* if the following holds: In any synchronous execution of $\mathcal{A}$ on a ring, in which all the identities are taken from I, and for any time $t < n/4$, at least one message is sent by some processor at time t. The following theorem is implicit in [2]:

Theorem 3.5 ([2], Lemma 5.1, Theorem 5.8) *Let $|I| \geq cn$, for $c > 1$. Then the average message complexity of any protocol which is active on I, where the average is taken over the worst case complexity over all rings with identities taken from I, is $\Omega(n \log n)$.*

Let I_0 be the set of all even ids, and I_1 be the set of all odd ids. Theorem 3.4 follows from Theorem 3.5 and the following Lemma:

Lemma 3.6 *Any consensus algorithm $\mathcal{A}$ is active either on I_0, or on I_1.*

Proof: (Outline) Otherwise, there exist two rings: R_0 with inputs from I_0 and R_1 with inputs from I_1, in which all the processors are quiescent at time $n/4-1$. Then, at time $n/4-1$ all processors in R_1 decide on 1, and all processors in R_0 decide on 0. Consider now a ring constructed by cutting and pasting half of the ring R_0 to half of R_1. Let P (Q resp.) be a processor in R which is surrounded by chains of $n/4-1$ processors from I_0 (I_1 resp.) on both sides. Since in a synchronous execution of less than $n/4$ time P (Q resp.) cannot distinguish between R and R_0 (R_1 resp.), in a synchronous execution of $\mathcal{A}$ on R, P will decide on 0 and Q will decide on 1,—a contradiction. ■

4 Two-processor bit complexity

Here we consider a chain of two processors connected by one link, and present exact or close to exact bit complexity bounds. Section 4.1 is devoted to achieving symmetry. In Section 4.2, we study breaking symmetry; we use for the lower bound an extension of the technique introduced in Section 4.1. In Section 4.3, we present a distributed computing analogue of the partition-into-rectangles technique of communication complexity [22, 21, 5], and use it to prove bit complexity lower bounds for `Leader` and some other problems.

4.1 Achieving symmetry with two processors

We start with a simple proof, which introduces some of ideas used later on in more sophisticated forms. These ideas are: consider specific schedulers (e.g., both processors wake up spontaneously), "cut-and-paste" two executions to produce a third one, and partitioning the set of ids according to the first bit sent by a processor. Recall that any lower bound on the complexity under any fixed scheduler is also a general lower bound on the complexity of a distributed algorithm/problem.

Theorem 4.1 *In chain of two processors with $M \geq 4$, $BitC($*`Consensus`*$) = 2$.*

Proof: To solve `Consensus` sending two bits, each processor sends the parity of its id to the other processor, and both decide on the (say) OR of these bits.

To prove the lower bound, consider an arbitrary algorithm solving `Consensus`. Let us fix any scheduler which wakes both processors spontaneously. Note that when a processor wakes up spontaneously, the first bit it sends (if any) is a function of its input only. Partition the set of ids into $S_0, S_1, S_\perp$, according to the first bit sent by a processor when it wakes up: $0, 1$, or nothing. Assume for contradiction that the bit complexity of `Consensus` is less than 2. Then $|S_0 \cup S_1| \leq 1$. Indeed, otherwise we can give two input ids from $S_0 \cup S_1$ to the processors, and in any execution where both processors wake up spontaneously, at least 2 bits are sent.

Therefore, $|S_\perp| \geq M - 1 \geq 3$ and hence, there are at least two even ids in $S_\perp$ or at least two odd ids in $S_\perp$. Assume w.l.o.g. that $x_0, y_0 \in S_\perp$ are even, and x_1, y_1 are odd. Then, in any execution where the inputs are $\langle x_0, y_0\rangle$, no bits are sent; so, both processors have to decide 0 immediately, without waiting to receive any bits. Hence, in any execution where the inputs are $\langle x_1, y_0\rangle$, also both have to decide 0, since y_0 decides so upon its wake-up. Moreover, x_1 decides also immediately, since the single bit, which may be sent, by our assumption, cannot be sent by y_0. Consider now the input pair $\langle x_1, y_1\rangle$. The decision must be 0, since x_1 decides so upon its wake-up, but this decision is not legal. ■

4.2 Breaking symmetry with two processors

Let us define a scheduler, for a chain of two processors, which will be used in some of our lower bound proofs. It is convenient to define the scheduler using real-time (recall that processors have no access to it).

Synchronous scheduler:

- Both processors wake up at the same time, say 0.
- At any integral time, every processor receives the first undelivered bit sent to it, if any.

The unique execution under the synchronous scheduler, implied for any input configuration, is henceforth called the *synchronous execution.*

We refer to the processors as A and B (the names A and B are not known to them). We represent any synchronous execution by its *synchronous history*, consisting of a sequence $a_1b_1a_2b_2...a_tb_t$ of values from $\{0, 1, \perp\}$. The bits received by A and B (if any), resp., at time i, are a_i, b_i, resp.; and if a_i or b_i is equal to $\perp$, no bit was received by A or B, resp. Note that $a_i = b_i = \perp$ is impossible for $i \leq t$ (at each step, at least one bit is sent). We denote by $h(x, y)$ the synchronous history corresponding to inputs (x, y). For a sequence h, let Inputs(h) denote the set of pairs (x, y) such that h is a prefix of $h(x, y)$. We call a prefix $a_1b_1a_2b_2...a_rb_r$ *symmetric* if $a_i = b_i \in \{0, 1\}$, $i = 1, 2, \ldots, r$.

Theorem 4.2 *In chain of two processors,* $2\lfloor\log_2 M\rfloor - 3 \leq BitC(\texttt{Leader}) \leq 2\lceil\log_2 M\rceil - 2$.

Proof: The following algorithm solves `MaxF`, and thus `Leader`, sending $2\lceil\log_2 M\rceil - 2$ bits: the two processors exchange their identities without their least significant bit; the larger abridged identity wins, and if they are equal, the identity with the unsent bit 1 wins.

In the rest of the proof, we establish the lower bound. Fix any reporting `Leader` algorithm, i.e. such that each processor sends its decision in its last message; thus, our lower bound may be off by two bits. We will show that there exists a synchronous history with a long *symmetric* prefix. Clearly, this will suffice, since a reporting `Leader` algorithm cannot terminate while the execution prefix is symmetric.

Let us refer to the processors as A and B. We construct a growing symmetric synchronous history prefix $h_i = a_1a_1a_2a_2 \ldots a_ia_i$, Inputs$(h_i) \neq \emptyset$, $i = 1, 2, \ldots$, as follows. Initially, h_0 is empty, and Inputs(h_0) =

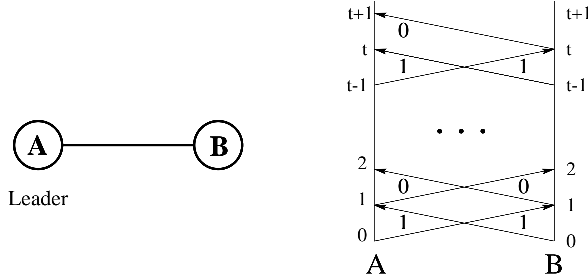

Figure 1: Illustration to the proof of Theorem 4.2.

$\{(x,y)|x,y \in Z_M, x \neq y\}$. For extending h_k to h_{k+1}, we restrict Inputs(h_k) so that for any input in the remaining part, the synchronous i-step execution: (i) ends with at least one bit in transit in each direction of the link, and (ii) the first undelivered bit is the same in both directions. Then, we extend h_k into h_{k+1} by delivering the next (equal) bits in the queues to both A and B.

As in the proof of Theorem 4.1, let us partition the set $S^0 = Z_M$ of all ids into $S^0_0, S^0_1, S^0_\perp$, according to what the first bit sent by a processor, when it wakes up spontaneously, is: $0, 1$ or nothing, respectively. Notice that there is at most one input in $S^0_\perp$. Indeed, if there are two inputs in $S^0_\perp$, we give one to A and the other to B, resulting in an execution, in which no bits are sent; this contradicts the assumption that the last bit sent is the decision.

Let S^1 be the largest set of S^0_0, S^0_1, and let a_1 be the corresponding bit sent. In the case $|S^1| \geq 2$, we fix $h_1 = a_1a_1$. Then, Inputs(h_1) $= \{(x,y)|x,y \in S^1, x \neq y\} \neq \emptyset$. When A and B get a pair of (distinct) ids from S^1, then the length 2 execution prefix has a synchronous history h_1: the same bit a_1 is delivered at time 1 to each of the processors. Note that in any execution which starts with an input from S^1, the same first bit is sent (it is possible that some additional bits are sent too).

The only new knowledge that each processor acquires up to the end of h_1, is that the input of the other processor belongs to the same subset S^1 as its own input. Hence, given that a synchronous history begins with h_1, the second bit sent by A or B depends *only* on its own input. As before, there are three subsets of S^1, denoted by $S^1_0, S^1_1, S^1_\perp$, according to the second bit sent (this bit could be sent either on wake up or after receiving the first bit a_1). Once more, $|S^1_\perp| \leq 1$. Let S^2 be the largest set of S^1_0, S^1_1, and let the corresponding bit sent be a_2. Then we set $h_2 = a_1a_1a_2a_2$ (see Figure 1). We continue in this way, while the next subset S^k contains at least two inputs; so long Inputs(h_k) $\neq \emptyset$.

We have that $|S^k| \geq \lceil (|S^{k-1}| - 1)/2 \rceil$, while $|S^0| = M$. It takes at least $\lfloor \log M \rfloor$ steps to get from an integer $r = M$ to a number smaller than 2, where in each step you replace r by $\lceil (r-1)/2 \rceil$. To see this, note that one such step maps the integers in the interval $[2^k, 2^{k+1}-1]$ onto the integers in the interval $[2^{k-1}, 2^k - 1]$ (indeed, $\lceil (2^{k+1}-1)/2 \rceil = 2^k$, $\lceil ((2^{k+1}-1)-1)/2 \rceil = 2^k - 1, \ldots, \lceil (2^k - 1)/2 \rceil = 2^{k-1}$, $\lceil ((2^k-1)-1)/2 \rceil = 2^{k-1} - 1$). Therefore, $|S^{\lfloor \log_2 M \rfloor - 1}| \geq 2$.

Let us give to A and B some pair of ids from $S^{\lfloor \log_2 M \rfloor - 1}$, and consider the synchronous execution for them. Since for each $i =$

$1, 2, \ldots, \lfloor \log_2 M \rfloor - 1$, two bits delivered to the processors are identical and since the last bits sent by the processors must be distinct, for a reporting algorithm, at least one more bit must be sent. That is, for the chosen input, at least $2 \lfloor \log_2 M \rfloor - 1$ bits are sent. We get the lower bound $2 \lfloor \log_2 M \rfloor - 1$ for reporting algorithms, and hence, the lower bound $2 \lfloor \log_2 M \rfloor - 3$ for the general case, as required. ■

Note that the gap between the upper and lower bounds given by Theorem 4.2 equals 1, when M is a power of 2, and equals 3, otherwise.

4.3 Two processor communication complexity approach

In this section, we build a distributed computing technique similar to the partition-into-rectangles technique of communication complexity [22, 21, 5], and use it for the two processor breaking symmetry problem. Instead of rectangles, we use "'semi-rectangles", since diagonal entries are undefined in the scenario we consider. As observed in the introduction, one difficulty in applying communication complexity lower bounds techniques like those of [22] to our model is that, unlike the model of [22], our model does not guarantee that there is a unique execution or even a unique output, for each given input vector. In this section, we use the synchronous scheduler to get such a unique execution.

4.3.1 Rectangle method for distributed computing

Recall that for any distributed algorithm $\mathcal{A}$ under a scheduler $\mathcal{S}$, any input I implies a unique execution $E(I) = E_{\mathcal{A},\mathcal{S}}(I)$ and thus a unique output *function* $f(I) = f_{\mathcal{A},\mathcal{S}}(I)$, whose values are the ordered tuples of the processors outputs. Consider the $M \times M$ matrix $D = \{(x, y) | \, x, y \in Z_M\}$. We fill its non-diagonal part by the (vector) values of $f(x, y)$, where x is given to A and y to B.

A set $R \subseteq D$ is called a *semi-rectangle* if there exist subsets $X, Y \subseteq Z_M$, such that $R = \{(x, y) | \, x \in X, \, y \in Y \text{ and } x \neq y\}$. It is easy to derive the following:

Proposition 4.3 *A set R is a semi-rectangle if and only if whenever $\{(x_1, y_1), (x_2, y_2)\} \subseteq R$ and $x_1 \neq y_2$, also $(x_1, y_2) \in R$.*

In the rest of Section 4.3, we consider the synchronous scheduler. Let $\text{Inputs}^0(h)$ denote the set of pairs $\{(x, y) | h(x, y) = h\}$. We suggest the following analogue of the rectangle partition statement of communication complexity (see [16, Section 1.2]):

Lemma 4.4 *For every synchronous history h, $\text{Inputs}^0(h)$ is a semi-rectangle.*

Proof: By Proposition 4.3, it suffices to show, for arbitrary pairs $(x_1, y_1), (x_2, y_2)$, s.t. $x_1 \neq y_2$, that if $h(x_1, y_1) = h(x_2, y_2) = h$, then also $h(x_1, y_2) = h$. Let $h = a_1 b_1 ... a_t b_t$, and $h(x_1, y_2) = c_1 d_1 c_2 d_2 ... c_t d_t$. Consider the execution $E(x_1, y_2)$. An easy induction shows that at any moment, A sees the same as at $E(x_1, y_1)$ and B sees the same as

at $E(x_2, y_2)$). Hence, $a_i = c_i$ and $b_i = d_i$ for all $1 \le i \le t$, and the decisions and halting moments are the same as well. ■

Let us assume now that the algorithm is reporting (processors send their decision at the end). This implies that processors A and B decide the same values in all executions with any fixed synchronous history h. So, the function f has the same value on the semi-rectangle $\text{Inputs}^0(h)$ of Lemma 4.4.

We say that a semi-rectangle R is *f-monochromatic* if f has the same value on all of R. Given a function f, let $D_f(M)$ be the minimum number of f-monochromatic semi-rectangles needed to partition the non-diagonal elements of D. We conclude from Lemma 4.4:

Corollary 4.5 *The number of distinct synchronous-histories, for a reporting algorithm, is bounded from below by $D_f(M)$.*

Theorem 4.6 *The bit complexity of any distributed algorithm for two processors, which under the syncronous scheduler computes a function f, is bounded from below by $\frac{1}{3}\log_2 D_f(M) - 2$.*

Proof: Given an algorithm, consider its extended reporting version. Each step in a synchronous history of this algorithm is represented by a pair (a_i, b_i), where (a_i, b_i) is one out of the eight pairs in $\{0, 1, \perp\}^2 \setminus \{(\perp, \perp)\}$. Hence, since by Corollary 4.5 there are at least $D_f(M)$ synchronous-histories, there must be a synchronous history h with at least $\log_8 D_f(M) = \frac{1}{3}\log_2 D_f(M)$ steps. The theorem follows since at each step at least one bit is sent, and only the two last decision bits sent by A and B are extra (reporting), as compared with the original algorithm. ■

4.3.2 Alternative lower bound proof for `Leader`

As an application of Theorem 4.6, we give a different proof of the $\Omega(\log M)$ lower bound on the bit complexity of breaking symmetry stated in Theorem 4.2. We say that a scheduler is *anonymous* if it is independent of the processor names; notice that the synchronous scheduler is anonymous. The following property is immediate:

Proposition 4.7 (mirroring) *For any anonymous scheduler and any pair of ids $x, y \in Z_M$, $x \neq y$:*
(i) *$E(y, x)$ is the execution obtained from $E(x, y)$ by permuting the names A and B everywhere, and*
(ii) *if $f(x, y) = (x', y')$, then $f(y, x) = (y', x')$.*

Consider the synchronous executions of any reporting `Leader` algorithm. By Proposition 4.7 and the specification of the `Leader` task, the output function f for `Leader` defined in Section 4.3.1 is *anti-symmetric*: $f(x, y) \neq f(y, x)$, for all legal inputs (x, y).

Theorem 4.8 *For an arbitrary anti-symmetric function f defined by the synchronous scheduler, $D_f(M) \ge M/2$.*

Proof: Let us consider a partition of D into $D_f(M)$ f-monochromatic semi-rectangles. Replace each semi-rectangle by the corresponding

rectangle (i.e., include the missing diagonal elements). Let $\mathcal{C}$ be the resulting *covering* of D by rectangles. Then the rectangles in $\mathcal{C}$ cover all non-diagonal entries, and, in addition, also certain k diagonal entries, for some $0 \leq k \leq M$.

Case 1: $k = M$ ($\mathcal{C}$ covers all the elements in $D = \{d_{ij}\}$). No rectangle in $\mathcal{C}$ can contain two distinct diagonal elements d_{ii}, d_{jj}, since $f(d_{ij}) \neq f(d_{ji})$. Thus, in this case $D_f(M) \geq M$.

Case 2: $k = 0$ ($\mathcal{C}$ does not cover any diagonal entry, and hence, is non-intersecting). We use a variant of the *rank method* ([16, Section 1.4]). For each rectangle R in $\mathcal{C}$, let A_R be the matrix in which the entries of R are 1 and the other entries are 0. Then $\sum A_R = J - I$, where J is the all-ones matrix and I is the identity matrix. Since $\text{rank}(J - I)$ is M, but $\text{rank}(A_R)$ is one, there are at least M matrices A_R. Hence, also in this case $D_f(M) \geq M$.

Case 3: $0 < k < M$. If $k \geq M/2$, use the result of Case 1 on the principal submatrix corresponding to these k indices. Otherwise, use Case 2 on the principal submatrix of the remaining $M - k$ indices. This implies $D_f \geq M/2$.

■

It follows from this theorem and Theorem 4.6, that $BitC(\texttt{Leader}) \geq \frac{1}{3}\log_2(M/2) - 2 = \Omega(\log M)$, as required.

Later we will use the following statement, which is implied by Theorem 4.8 and Corollary 4.5.

Lemma 4.9 *The number of distinct synchronous-histories of any reporting* `Leader` *algorithm for two processors is at least $M/2$.*

5 Easy cases for a chain

We now consider the chain topology, with $n+1$ processors and n links. Most of the related exact bounds are developed in the sibling paper [8]. Sections 5.1 and 5.2 present main insights and results from there, while Section 5.3 adds one more result. We thus isolate the interesting hard cases to be treated in Section 6.

5.1 `Consensus`

A $2n$ algorithm for `Consensus` is simple: If an intermediate processor wakes up spontaneously, it sends nothing. When a terminal wakes up, it broadcasts its parity. Any processor decides on OR of the two bits it has received, in its id and/or messages.

One would expect that both $BitC(\texttt{Consensus})$ and $BitC(\texttt{Leader})$ are at least n, because if less than n bits are sent, the chain would be divided into two parts with no information exchange between them. The exact bound for `Consensus` is $2n$, by the following statement:

Theorem 5.1 ([8], Theorem 3.2) *For a chain with n links, the message complexity of* `Consensus` *is at least $2n$. 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. These bounds for M are tight.*

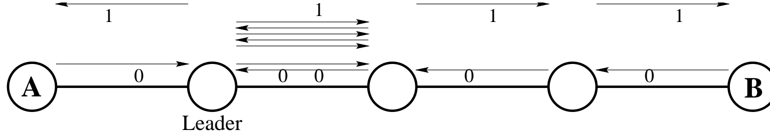

Figure 2: Execution of the algorithm for `Leader` in a chain.

5.2 `Leader`: even length chain

Recall that the chain length is defined as the number of links in it. In the case when the chain length is known to be even, a $2n$ algorithm (which does not use ids!) is as follows. If an intermediate processor wakes up spontaneously, it sends nothing. When a terminal wakes up, it propagates the sequence $0101\ldots$. If the two waves with these sequences meet at some processor, it is the leader. If they meet on an edge e, necessarily one bit sent on e is 1 and the bit sent in the opposite direction is 0; the processor that has transmitted 1 and received 0 is the leader. Also, 1 is broadcasted to all other processors to indicate that they are not leaders. Another algorithm, with bit complexity $1.5n$, is given in [8].

These algorithms, together with the above common sense observation, imply that for an even chain, $BitC(\texttt{Leader}) = \Theta(n)$. The exact bound is $1.5n$, as is implied by the two following results:

Theorem 5.2 ([8], Theorem 5.2) *For 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.*

Theorem 5.3 ([9], Theorem 2) *For $M \geq 6$, $BitC(\texttt{Leader})$ in an even length chain is 0 if $n = 2$, 4 if $n = 4$, 8 if $n = 6$, and $1.5n$ if $n \geq 8$, if inputs are given to the terminals only and even if the value of n is known to the processors. Moreover, for $n \geq 10$, the algorithm given in [8] is the single optimal algorithm.*

5.3 `Leader`: general chain

A $2(n + \log M - 1)$ algorithm is as follows. Similarly to the even case, two $000\ldots$ sequences are propagated from the terminals. If the two waves with these sequences meet at some processor, it is the leader. If the waves meet on an edge e, then the two processors at the endpoints of e solve `Leader` with their identities as inputs. Also, 1 is broadcasted from the meeting place to inform the rest of processors that they are not leaders (for illustration see Figure 2).

Let us show the lower bound $\Omega(\log(M/n))$ for an odd length case (and thus, for the general one). Call a *super-input* the tuple of $(n+1)/2$ ids in the chain from a terminal processor inward; thus, each instance of `Leader` has two super-inputs, from the sides of the two terminals. Let $\tilde{M}$ denote $\lfloor 2M/(n+1) \rfloor$. We restrict the problem to the super-inputs $i_0 = (1, 2, \ldots, (n+1)/2)$, $i_1 = ((n+1)/2+1, (n+1)/2+2, \ldots, n+1)$, $\ldots$, $i_{\tilde{M}-1} = ((\tilde{M}-1)(n+1)/2+1, (\tilde{M}-1)(n+1)/2+2, \ldots, \tilde{M}(n+1)/2)$. Let us, given an algorithm $\mathcal{A}$, replace each half-chain by a single *super-processor*, which given input $q \in [0..(\tilde{M}-1)]$, simulates in its local

memory the computation of $\mathcal{A}$ in this half-chain, where the processors are given the respective components of i_q as their inputs. In particular, it decides "leader" if some processor in its half-chain decides "leader" according to $\mathcal{A}$, and decides "non-leader" if all processors in its half-chain decide so according to $\mathcal{A}$. Thus we obtain a two-processor `Leader` algorithm, $\mathcal{A}'$, over the middle edge. Observe that any pair of distinct super-inputs corresponds to a set of distinct original inputs in the entire chain, and thus to a legal instance of the original problem. Therefore, $\mathcal{A}$ decides "leader" at exactly one processor, which implies that $\mathcal{A}'$ decides properly. Thus, $\mathcal{A}'$ is a valid two-processor `Leader` algorithm, with $\tilde{M} = \lfloor 2M/(n+1) \rfloor$ possible inputs. By Theorems 4.2 and 4.6, its bit complexity, and thus the bit complexity of $\mathcal{A}$, is $\Omega(\log(M/n))$, as required.

Together with the bound $\Omega(n)$, of our common sense observation and of [8, Theorem 5.2], this bound implies: $BitC(\texttt{Leader}) = \Omega(n + \log(M/n)) = \Omega(n + \log M - \log n) = \Omega(n + \log M)$. Summarizing, we obtain:

Theorem 5.4 *For the chain topology,* $BitC(\texttt{Leader}) = \Theta(n + \log M)$.

6 Breaking Symmetry in an Odd Length Two-Input Chain

As we have seen in the previous section, the easy cases for `Leader` in a chain are when there is a middle processor, or when every processor has a distinct input. In the following two subsections, we consider the case when the length of the chain is odd and only the two terminals have inputs. We assume that the terminal processors, and only them wake up spontaneously. This scenario is analogous to the linear array problem studied in the context of communication complexity [5, 14, 21].

An obvious $O(n \log(M))$ bit complexity algorithm for `Leader`, and moreover, for `MaxF`, is as follows: Both terminals relay their ids through the chain; each processor learns on the placement of the maximal id when both waves reach it. To prove the matching lower bound of $\Omega(n \log(M))$, we use a variant of Dietzfelbinger's [5] proof strategy for the communication complexity model. In Section 6.1, we describe and analyze our main tool, the outside precedence scheduler; it is applied to `Leader` in Section 6.2. In Section 6.3, we show that for `MaxF`, there is no scheduler that can reduce the bit complexity below $\Omega(n \log(M))$.

6.1 Outside-Precedence Scheduler

We denote the middle link by $\tilde{e}$, and the terminal processors by A and B. The part of the chain between A and $\tilde{e}$ is called the *left part of the chain*, and its part between $\tilde{e}$ and B is called the *right part of the chain* (see Figure 3). We say that a part of the chain is *quiescent*, at some time, if none of the links in this part has messages in transit.

We proceed to define the *Outside-Precedence Scheduler* (OPS). For a distributed algorithm, its execution defined by OPS is composed of consecutive *rounds.* Each round begins with a *right phase*, where bits are delivered on the links in the right half of the chain, until this area

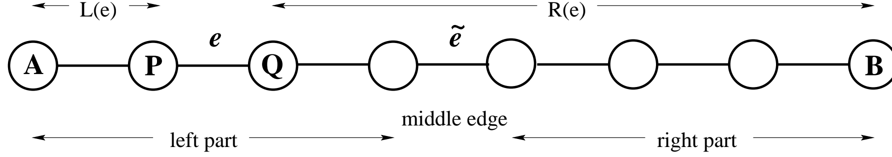

Figure 3: Illustration to notation for the odd length two-input chain.

becomes quiescent. Then, there is the *left phase*, defined similarly for the left half of the chain. At the end of the round, when both halves are quiescent, the *middle phase* is executed, where the (single) first bit in transit, if any, is delivered along each one of the directions on $\tilde{e}$. It should be noted that over the middle edge, the OPS-execution is similar to the synchronous execution defined in Section 4.2, and hence the *synchronous history* (or *s-history*, for short) of $\tilde{e}$ can be defined by the same notation. The first right (left) phase starts by waking up the right (left) terminal. At each step of a right or left phase, a single bit is delivered, hence a single processor becomes active. OPS decides on the link and direction on it, where the first bit will be delivered next, by the following "outside preference" rules:

- Links farther away from $\tilde{e}$ are preferred to links closer to it.
- For each link, the direction away from $\tilde{e}$ is preferred to the direction towards $\tilde{e}$.

In what follows, up to the end of Section 6.2, let an arbitrary distributed algorithm, $\mathcal{A}$, be fixed. For any input, OPS defines a unique execution, for it; we denote the OPS-execution with inputs x and y given to A and B, respectively, by $OPS(x,y)$.

Let $e = (P,Q) \neq \tilde{e}$, be a link, where P is its left end-node and Q is its right end-node. We define the (*plain*) *history* $h_e(x,y)$ of e during $OPS(x,y)$, as the sequence of all sending and delivering events on it, in the chronological order. Let $L(e)$ and $R(e)$ denote the sets of processors to the left and to the right of e, respectively. We denote by $OPS_{L(e)}(x,y)$ (resp., $OPS_{R(e)}(x,y)$) the subsequence of steps in $OPS(x,y)$ where processors in $L(e)$ (resp., $R(e)$) are active. This subsequence contains all activity on the links to the left (resp., right) of e, and also sending messages on e by P (resp., Q) and delivering messages on e to P (resp., Q).

The following property is crucial for our lower bound proof presented in the next section. We use the notion of *PQ-queue* (resp., *QP*-queue) as the messages in transit on $e = (P,Q)$ from P to Q (resp., from Q to P).

Lemma 6.1 (Cut-and-Paste Property) *Let e be a link distinct from $\tilde{e}$, and (x,y) and (x',y') be two input pairs. If the histories $h_e(x,y)$ and $h_e(x',y')$ coincide, then:*

1. $h_e(x,y') = h_e(x,y) = h_e(x',y')$,
2. $OPS_{L(e)}(x,y') = OPS_{L(e)}(x,y)$,
3. $OPS_{R(e)}(x,y') = OPS_{R(e)}(x',y')$.

Proof: Consider first the case when e lies in the left half of the chain.

In this case, the entire $L(e)$, together with e, is contained in the left half of the chain. Let e be (P, Q).

Consider an arbitrary OPS-execution. Divide the sequence of its execution steps into the interleaving intervals of continuous activity in $R(e)$ and in $L(e)$, called henceforth the *right$_e$* and *left$_e$ intervals*, respectively. By the phase structure and outside preference of an OPS-execution, the first interval starts from waking up B and delivering the first bit sent by B, if any, continues in the right half of the chain, and finishes when the entire right half becomes quiescent. The second interval starts from waking up A and delivering the first bit sent by A, if any, continues in $L(e)$, and finishes when the entire $L(e)$ becomes quiescent. Notice that at its finishing moment the QP-queue is empty, since Q has not been active yet. Clearly, the contents of the considered intervals are mutually independent.

Claim 6.2

(a) *An OPS-execution is active in $R(e)$ if and only if all queues in $L(e)$ and the QP-queue are quiescent, except for maybe the first right$_e$ interval.*

(b) *The transition from any non-first right$_e$ interval to the next left$_e$ interval happens when and only when Q is active and at least one message is in transit from it to P; then, at the next step, the first one of such messages is delivered to P.*

(c) *The execution at any right$_e$ interval, I, including its finishing moment, depends only on the prefix of $OPS_{R(e)}$ before I and on the state of the PQ-queue at the beginning of I.*

(d) *The execution at any left$_e$ interval, J, including its finishing moment, depends only on the prefix of $OPS_{L(e)}$ before J and on the state of the QP-queue at the beginning of J.*

Proof:
(a) Denote the set of queues mentioned in **(a)** by $\mathcal{L}$. Obviously, if all queues in $\mathcal{L}$ are quiescent, then the coming execution step belongs to a right$_e$ interval. The other direction: Assume that at least one of the queues in $\mathcal{L}$ is non-empty. By the phase structure of OPS, this can happen only in a left phase, except for maybe the first right$_e$ interval. By the outside preference, during any left phase, OPS continues to deliver in $L(e)$, while there is something to deliver.

(b) When Q is active, it is a left phase and a right interval, by definition. Immediately after any step where Q sends to P, by the outside preference of OPS in a left phase, the first such message sent is delivered to P, and thus the next left$_e$ interval starts. For the "only when" direction, observe that, by **(a),** during any non-first right$_e$ interval, all queues in $\mathcal{L}$ are quiescent. The only way to violate this property is to send a message from Q to P.

(c) During any right$_e$ interval I, $L(e)$ is quiescent. Hence, the only external influence w.r.t. $R(e)$, during I, is delivering from the PQ-queue. Besides, during a right interval, the PQ-queue is not appended. The finishing moment is either sending from Q to P, which is an internally defined event, or the end of execution. The latter event happens when $R(e)$ becomes quiescent, which is also an internally defined event.

(d) During any left$_e$ interval, while there are non-empty queues in $\mathcal{L}$, the behavior of OPS depends only on these queues, by the outside preference rules of OPS. Moreover, no external information w.r.t. $L(e)$ can influence this part of the execution, except for delivering from the QP-queue. Besides, during a left interval, the QP-queue is not appended. The interval finishes when all queues in $\mathcal{L}$ become empty, which is an internally defined event. ■

Fix some inputs t, w, and denote by h_e the history of e, $h_e(t, w)$, for short. We analyze now how the above partition of $OPS(t, w)$ into intervals determines a similar partition of the events in h_e.

Claim 6.3 *The partition of h_e induced by the right$_e$ and left$_e$ intervals is defined uniquely by h_e.*

Proof: Notice first that the first right$_e$ interval contributes nothing to h_e. The events on h_e which correspond to the first left$_e$ interval (started by waking up A) consist of a (possibly empty) sequence of PQ-sending events which precede the first QP-sending event on h_e.

We show now that to each non-first left$_e$ interval, there corresponds a non-empty subsequence of events in h_e of a certain type. By Claim 6.2(b), each left$_e$ interval, except the first one, starts with a QP-delivering of a single bit which is preceded by a QP-sending of one or more bits, and vice versa: each QP-delivering of a single bit which is preceded by QP-sending of one or more bits, is the first event of a left$_e$ interval. Moreover, the subsequence of h_e which corresponds to each such left$_e$ interval is the maximal (by inclusion) subsequence of h_e which starts with a QP-delivering of the abovementioned type and includes only QP-deliverings and PQ-sendings. Note that all pieces of h_e described above are well defined given h_e only.

Finally, observe that the non-empty subsequences of h_e which correspond to the left$_e$ intervals, split the remaining events in h_e into non-empty subsequences corresponding to non-first right$_e$ intervals, in a natural way. ■

Corollary 6.4 *For any number i, the state of the QP- and PQ-queues after finishing of the ith right$_e$ or i^{th} left$_e$ interval is defined uniquely by h_e.*

Indeed, any such queue state can be computed on the basis of the corresponding prefix of h_e, while this prefix, in its turn, is defined uniquely by h_e, by Claim 6.3.

Now, we are ready to proceed to the proof of the lemma, i.e., of the Cut-and-Paste property. For any i, we assign to the prefix of i first right$_e$ intervals at $OPS_{L(e)}(x, y')$ the prefix of i first right$_e$ intervals at $OPS_{L(e)}(x', y')$, and similarly for the prefixes of i first left$_e$ intervals at $OPS_{L(e)}(x, y')$ and $OPS_{L(e)}(x, y)$. We will prove that the corresponding prefixes coincide, by induction on i.

Basis: Since the first right$_e$ interval and the first left$_e$ interval depend on the inputs only, the first right$_e$ intervals are the same at $OPS(x, y')$ and at $OPS(x', y')$, and the first left$_e$ intervals are the same at $OPS(x, y')$ and at $OPS(x, y)$. That is, the statement of the lemma holds for the execution prefix covered by them. As a result, the corresponding prefixes of the history of e are the same.

Assumption: We assume that for some prefix of the sequence of intervals at $OPS(x, y')$, the statement of the lemma holds for any of its sub-prefix of right_e or left_e intervals.

Induction Step: We now consider the next interval. Let it be the right_e one. Consider its beginning. The states of all processors in $R(e)$, as well as of all queues between them are the same at $OPS(x, y')$ and $OPS(x', y')$, since they are defined by the same preceding execution, by the induction hypothesis for right_e intervals. The state of the QP- and PQ-queues is the same by the induction hypothesis and Corollary 6.4. By Claim 6.2(c), the executions in the next right_e intervals at $OPS(x, y')$ and $OPS(x', y')$ go in the same way and finish after the same step. This implies the same continuation of $h_e(x, y')$ as in $h_e(x', y') = h_e(x, y)$, as required. The case when the next interval is the left_e one is analyzed similarly, using Claim 6.2(d).

This is the end of the proof of Lemma 6.1, for the case when e is to the left of $\tilde{e}$. The proof for the case when e is at the right chain half, is similar. ■

Let e_1, e_2 be edges belonging to different half-chains, and let h_1, h_2 be their respective histories in a given execution. Then the *bracket history* of (e_1, e_2) is the ordered pair (h_1, h_2). The following analogue of the Bracket Lemma [5, Lemma 2.5] holds:

Lemma 6.5 (Bracket Lemma) *If two OPS-executions of $\mathcal{A}$ have distinct s-histories on $\tilde{e}$, then the bracket histories of (e_1, e_2) in these executions are also distinct.*

Proof: Assume for a contradiction that $\text{OPS}(x, y)$ and the $\text{OPS}(x', y')$ have different s-histories h, h' in $\tilde{e}$, respectively, but both executions have the same bracket history (h_1, h_2) on (e_1, e_2). Assume, w.l.o.g., that e_1 is closer to the terminal given input x. Then, by Lemma 6.1 applied to edge e_1, $\text{OPS}(x, y')$ has the same bracket history (h_1, h_2) on (e_1, e_2), as well as the same s-history h' on $\tilde{e}$, as $\text{OPS}(x', y')$. Repeating for inputs (x, y') and (x, y), and edge e_2, we arrive at an OPS-execution of $\mathcal{A}$ for inputs (x, y) which also has the s-history h' on $\tilde{e}$. Thus we get two different executions for (x, y), one where $\tilde{e}$ has s-history h and the other where it has s-history h', contradicting uniqueness of the OPS-execution. ■

6.2 Bit complexity of `Leader` in the two-inputs case

Consider a chain of an odd length where inputs are given only at the terminal processors. Here we use results of the previous section in order to prove that $BitC(\texttt{Leader}) = \Omega(n \log M)$.

Consider the notion of a reporting algorithm (see Section 2), for the setting where only terminal processors decide: we require that the terminals relay their decisions one to the other, along the chain. Fix an arbitrary `Leader` algorithm, $\mathcal{A}$, for an odd length chain with inputs at the terminals, and consider its reporting extension, $\mathcal{A}'$. The extra cost of $\mathcal{A}'$, w.r.t. $\mathcal{A}$, is $2n$ bits.

Lemma 6.6 *The number of distinct s-histories of $\mathcal{A}'$ under OPS on the middle link $\tilde{e}$, over all pairs of inputs, is at least $M/2$.*

Proof: Similarly to Section 5.3, replace each half-chain by a single super-processor which simulates in its local memory the computation of $\mathcal{A}'$ under OPS in this half-chain. Thus, one computation step of a super-processor corresponds to one phase of a computation in the corresponding half chain, which starts when the (quiescent) half chain receives a message from outside, and terminates when it becomes quiescent again. We get a reporting two-processors `Leader` algorithm, $\mathcal{B}'$, over $\tilde{e}$, and its execution under the synchronous scheduler (defined in Section 4.2), such that for each input (x, y), the messages exchanged on $\tilde{e}$ are the same as in the (unique) outside-precedence execution of $\mathcal{A}'$ on this input. Note that this transformation is possible since the chain is of odd length, and hence the network is split into two symmetric parts, preserving the requirement that the two processors can be distinguished only by their identities. The result follows by Lemma 4.9. ■

Theorem 6.7 *In an odd length chain with inputs only at the terminals,* $BitC(\texttt{Leader}) = \Theta(n \log M)$.

Proof: For an arbitrary algorithm $\mathcal{A}$ solving the problem, consider first its reporting extension $\mathcal{A}'$. As a consequence of Lemma 6.6, there are w, $w \geq M/2$, different inputs that lead to w different s-histories of $\mathcal{A}'$ on $\tilde{e}$. Let the inputs be $(x_1, y_1), \ldots, (x_w, y_w)$, and the s-histories on $\tilde{e}$ be $\tilde{h}(x_1, y_1), \ldots, \tilde{h}(x_w, y_w)$. By Lemma 6.5, these w inputs lead to different bracket histories at any pair of symmetric links (i.e., links at the same distance k from the terminals). Let the bracket history at some symmetric distance k, when the input is (x_i, y_i), be $(h_k, h_{n-k+1})(x_i, y_i)$. An s-history over $\tilde{e}$ is a word over the 8 letters alphabet $\{0, 1, \perp\}^2 \setminus \{(\perp, \perp)\}$, whose length is at most twice the number of bits sent in it. Also, a bracket history is represented by a word over the 5 letters alphabet $(\{0, 1\} \times \{L, R\}) \cup \{,\}$, whose length is greater by 1 than the number of bits sent in it. Hence, any lower bound of the kind $O(\cdot)$ on the sum of history lengths would imply the same $O(\cdot)$ lower bound on the number of bits sent. The details are as follows.

The total number of bits sent in the execution on input (x_i, y_i) is at least:

$$\frac{1}{2}|\tilde{h}(x_i, y_i)| + \sum_{1 \leq j \leq (n-1)/2} (|(h_j, h_{n-j+1})(x_i, y_i)| - 1) \ .$$

The maximum value of the above sum is at least its average over the w such histories:

$$\frac{1}{2} \cdot \frac{1}{w} \sum_{1 \leq i \leq w} |\tilde{h}(x_i, y_i)| + \sum_{1 \leq j \leq (n-1)/2} (\frac{1}{w} \sum_{1 \leq i \leq w} |(h_j, h_{n-j+1})(x_i, y_i)| - 1) \ .$$

The average length of $w \geq M/2$ distinct words over size c alphabet, for $c = 8$ or $c = 5$, is at least $\log_c w - const_1 \geq \log_c M - const_2$. Hence, $BitC(\mathcal{A}') \geq \Omega(\frac{1}{2}(\log_8 M - const_2) + \frac{(n-1)}{2}(\log_5 M - const_2 - 1)) = \Omega(n \log M)$. The additional cost of the reporting $2n$ bits does not affect this asymptotic growth, so $BitC(\mathcal{A}) = \Omega(n \log M)$ as well. By the obvious algorithm for `Leader` described at the beginning of the section, $BitC(\texttt{Leader}) = O(n \log M)$, and the Theorem follows. ■

6.3 Best scheduler complexity of MaxF

There exist Leader algorithms for a chain of odd length with inputs only at the terminal processors which have low bit complexity in some executions. Consider an arbitrary valid Leader algorithm $\mathcal{A}$, and add to it the following preprocessing stage. When a terminal wakes up spontaneously, it sends bit 1 to the other terminal and waits for a response. If a terminal is waken up by a message, it decides that it is not the leader, sends 0 to the other terminal, and stops. In any case, all intermediate processors relay the first bit in either direction. If a terminal receives 0 as the response to the first bit sent, it decides to be the leader and stops. If a terminal receives 1 as the response, it begins to behave according to algorithm $\mathcal{A}$. It is easy to see that the extended algorithm solves Leader. Moreover, its complexity under any scheduler that wakes up spontaneously only one of the terminals is $2n$. This is less than the worst case lower bound of $\Theta(n \log M)$ in Theorem 6.7.

In this section, we show that this is not the case for MaxF. Specifically, we show that for any reporting algorithm $\mathcal{A}$ for MaxF in a chain, there exists an input, such that in *any* execution of $\mathcal{A}$ with this input, $\Omega(n \log M)$ bits are sent. That is, no "benevolent" scheduler can reduce the bit complexity of this algorithm on this specific input below the worst case complexity of MaxF. As a consequence, no scheduler can reduce the $\Omega(n \log M)$ bit complexity of MaxF.

Working with functions (instead of tasks) allows to strengthen Lemma 4.4, as in Lemma 6.8 below. We begin with giving some definitions and notation. Let us call A and B the end processors of a chain, which compute a function $f(x, y)$. Assume that $f(x, y)$ is a boolean value to be decided by A, when A has input x and B has input y; B can either decide the same value or its complement. For MaxF, $f(x, y)$ is 1 if $x > y$, and 0 if $x < y$.

Consider an arbitrary reporting algorithm $\mathcal{A}$, which computes a function $f(x, y)$ of inputs (x, y), with $x \neq y$. Let us fill the non-diagonal entries of the output matrix D (of dimension $M \times M$) with the values of f. To facilitate the analysis, assume that in each execution of $\mathcal{A}$, at most one bit is delivered on any link at any given moment. Then, the history of a link e can be represented by the bit sequence $d_1 b_1 d_2 b_2 \dots d_t b_t$, where d_i is L(eft) or R(ight), and b_i is a bit, with the obvious meaning: b_i is the i-th bit delivered on e, and d_i is the direction to which b_i was delivered. Let Inputs(e, h) be defined as the set of all inputs (x, y), s.t. there *exists* an execution of $\mathcal{A}$ on input (x, y) where the history at e is h. Recall that a semi-rectangle is called f-monochromatic if all its entries in D are the same.

Lemma 6.8 *Let h be a history on a link e. Then, Inputs(e, h) is an f-monochromatic semi-rectangle.*

Proof: To show that Inputs(e, h) is a semi-rectangle we prove that if $(x_1, y_1), (x_2, y_2) \in$ Inputs(e, h), then also $(x_1, y_2) \in$ Inputs(e, h). Let E_1 and E_2 be two executions of $\mathcal{A}$ on inputs (x_1, y_1), (x_2, y_2), resp., for which the history at e is h. The construction of the third execution, E, for (x_1, y_2), is done by the standard cut-and-paste technique, which simulates E_1 (on input (x_1, y_1)) on the left side of e, and the execution E_2 (on input (x_2, y_2)) on the right side of e, and both of them on e. Then, in E, the processors to the left (resp., right) of e will not see any

difference from working in E_1 (resp., in E_2), so their state sequences will agree with the algorithm. In other words, E will be an execution of $\mathcal{A}$ on (x_1, y_2), with history h on e, as required (notice that we even did not use that two given inputs are distinct).

Since the algorithm is reporting, the last bits in h sent in each direction on e are the decisions of terminals, i.e., the function value is fixed by h. Hence, the rectangle Inputs(e, h) is f-monochromatic. ∎

Remark: Even if E_1 and E_2 are defined by the same scheduler, say $\mathcal{S}$, E may be not defined by $\mathcal{S}$.

For an input (x, y) and a link e, we denote by $h^*_{x,y}(e)$ the shortest possible history on e produced by any execution of $\mathcal{A}$ on input (x, y) (ties are broken in an arbitrary manner). Clearly, $\sum_e |h^*_{x,y}(e)|$ is a lower bound on the sum of the lengths of link-histories of any execution of $\mathcal{A}$ on input (x, y).

A subset I of non-diagonal elements of the output matrix D will be called an *f-fooling set* (as in [16]), if no f-monochromatic semi-rectangle contains two elements from I.

Lemma 6.9 *Let I be a fooling set and e be any link. Then the average length of $h^*_{x,y}(e)$, taken over all input pairs (x, y) in the fooling set I, is at least $\lfloor \log |I| \rfloor - 1$.*

Proof: By Lemma 6.8, for each history h, Inputs(e, h) is an f-monochromatic rectangle, and hence it contains at most one element from I. Thus the considered histories $\{h^*_{x,y}(e) | (x, y) \in I\}$ are pairwise distinct. The corollary follows since the average length of ℓ distinct binary sequences is at least $\lfloor \log \ell \rfloor - 1$. ∎

Let $fool(f)$ denote the maximal cardinality of a fooling set for f, and let I be a fooling set of this cardinality. Note that the number of bits sent in a computation is half the length of the history of this computation, at any link. Hence for any link e, by Lemma 6.9, the average of the number of bits in $h^*_{x,y}(e)$ taken over the inputs from I, is at least $(\lfloor \log fool(f) \rfloor - 1)/2$. Summing over all links, we get that the average, over all inputs in I, of the total number of bits sent over all the n links, in any execution of $\mathcal{A}$ on such an input, is at least n times that number. This gives the following result:

Theorem 6.10 *For any algorithm $\mathcal{A}$ computing a function f, there exists an input (x, y), such that in any execution of $\mathcal{A}$ on this input, at least $n(\lfloor \log fool(f) \rfloor - 1)/2$ bits are sent.*

We now turn to the function corresponding to `MaxF`. It equals 1 below the diagonal of the output matrix D and 0 above it. Hence, each rectangle Inputs(e, h) is either entirely not below the diagonal or entirely not above it. It can be easily checked that the set of $\lfloor M/2 \rfloor$ sub-diagonal elements $\{D[2i, 2i-1] | 1 \leq i \leq M/2\}$ of D is a fooling set for `MaxF` (see Figure 4). Indeed, for any pair of matrix elements from this set, the 2×2 rectangle defined by them contains three elements under the diagonal and one above it; hence, such a pair cannot be contained in an f-monochromatic semi-rectangle. Therefore, $fool($`MaxF`$)$ is at least $\lfloor M/2 \rfloor$, and Theorem 6.10 implies:

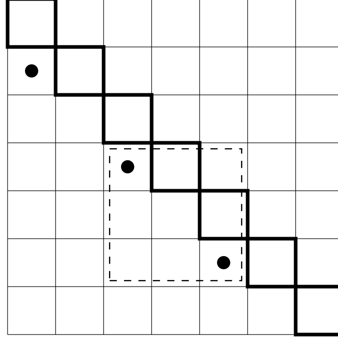

Figure 4: Fooling set for MaxF.

Corollary 6.11 *For any algorithm $\mathcal{A}$ for* MaxF*, there exists a legal input (x, y) (i.e., $x \neq y$), such that in* any *execution of $\mathcal{A}$ on this input, at least $n(\lfloor \log M \rfloor - 2)/2$ bits are sent.*

As a consequence, the *best scheduler complexity* of MaxF is $\Omega(n \log M)$ (see Section 2 for the definition). [In fact, the statement of Corollary 6.11 is *equivalent* to this consequence. At the first glance, the statement: *for each algorithm there exists an input s.t. each execution sends many bits*, looks stronger than the statement: *for each algorithm and each fixed scheduler there exists an input s.t. the corresponding execution sends many bits*, since the "worst" inputs for different schedulers must not be the same. However, since we do not restrict the knowledge of a scheduler, there exists the *totally best* scheduler, over all inputs. Indeed, such a scheduler is able to check, what is the input given, and simulate the scheduler which is the best w.r.t. this input.]

In fact, the above proofs work also in the setting where *all* processors are given inputs (this observation is joint with Noam Solomon), as follows:

Corollary 6.12 *The best scheduler bit complexity of* MaxF *is $\Omega(n \log M)$, in the setting when all processors are given inputs and if $n \leq (1-c)M$, for a constant $c > 0$. Therefore, in this case, $BitC$(*MaxF*) $= \Theta(n \log M)$.*

Proof: Let us give to all $n-1$ intermediate processors *fixed* distinct inputs from the range $[1, n-1]$, and to the terminals, various inputs from the range $[n, M]$. Thus we define a problem similar to the considered above in this section, where the two terminals are given inputs from the set $Z_{M'} + n - 1 = \{i + n - 1 : i \in Z_{M'}\}$, where $M' = M - n + 1 = \Omega(M)$. It is easy to see that fixed inputs at the intermediate processors and assuming inputs from $Z_{M'} + n - 1$ rather than $Z_{M'}$ do not prevent the above proofs to be valid. This implies the $\Omega(n \log M') = \Omega(n \log M)$ bound, as required. ■

The following observation is worth to mention. We call a binary function $f(x, y)$ *anti-symmetric* if for every $x \neq y$, $f(x, y) = 1 - f(y, x)$. Note that MaxF is a special case of an anti-symmetric function if completed for the diagonal elements in an arbitrary way. Consider the traditional communication complexity model in which the processors have

distinct identities A and B which are mutually known, and all inputs from $Z_M \times Z_M$ are allowed. For this model, our lower bound of Corollary 6.11 for `MaxF` can be generalized (with a slightly sharper bound) to hold for any anti-symmetric function $f(x, y)$, as follows. Since in this model the inputs can be identical, the M diagonal elements of D are legal inputs, which, by the definition, form a fooling set for any anti-symmetric function. Thus, by using this fooling set in the above proof, our lower bound extends to any anti-symmetric function, in this model. An interesting open problem is whether a similar lower bound for an arbitrary anti-symmetric function applies also to our model, in which the diagonal elements are not a fooling set (since identical inputs are not allowed).

7 Rings

Theorem 7.1 *In a ring,* $BitC(\texttt{Leader}) = \Omega(n \log M)$.

Proof: The proof is similar to the one of Theorem 4.2; we do not describe once more some details presented there.

Consider an arbitrary reporting algorithm for `Leader` in a ring (i.e., such that the last bit sent by any processor, at any outcoming link, is its decision), and its synchronous executions. For the theorem statement, it does not matter that the lower bound for reporting algorithms may be off by $2n$ bits as compared with the general bound. In the proof of the lower bound, we assume that the processors are arranged on the ring in a symmetric way, so that the first link at each processor is oriented clockwise. Let us analyze the algorithm steps.

First step: When a processor wakes up, at time 0, there are 5 possibilities for the first sending operation: Either no bit is sent ($\perp$), or the first sent bit is 0 or 1, clockwise or counter-clockwise (though the algorithm is aware of local link numbering only, our assumption allows us to state so). Thus the input set Z_M is partitioned into the five corresponding subsets. We choose the largest one; denote it by S_1. Provided its size is at least n, we give inputs from S_1 to all processors.

Observe that the size of the subset corresponding to $\perp$ must be less than n. Indeed, otherwise we can give inputs from this subset to all the processors, and then the algorithm terminates before processors sent their decisions bits, contradicting the reporting property.

Second step: Since all the inputs are from S_1, each processor receives, at time 1, the same bit from the same side, and that bit is the same as it sent. Hence, its state depends on its input only. We choose a subset S_2 in a similar way.

We continue with such a construction as long as $|S_k| \geq n$.

Thus, for any $k \geq 2$, $|S_k| \geq \lceil (|S_{k-1}| - n + 1)/4 \rceil$, where $|S^0| = M$. We get that there are $\Omega(\log(M/n))$ steps, and thus $\Omega(n \log(M/n))$ bits are sent. Combining this with the known $\Omega(n \log n)$ lower bound (implied by the message complexity result of [4]) we get the required bound: $\Omega(n \log(M/n)) + \Omega(n \log(n)) = \Omega(n \log M)$. ■

8 Conclusions and Discussion

We have considered three fundamental distributed computing problems: Consensus, Leader and MaxF, and shown that they have the same message complexity in chains and rings. However, we showed that while a solution to Leader can be very useful to solve Consensus, the opposite is not true. Finding similar relations between other pairs of distributed tasks can be interesting in general.

Then, we proceeded to study the problems under a finer measure of complexity: the number of bits sent. Our study uncovered interesting relations between communication complexity and distributed computing, two areas that were studied independently in the past.

In particular, the symmetric execution method can be viewed as a dual of the partition-into-rectangles technique. It shows that certain short symmetric executions define each a square $S \times S$, for some *large* set $S \subseteq Z_M$, s.t. all pairs of distinct inputs from $S \times S$ have the same history. It is worth to mention that we enhance, by this method, the rectangle method of communication complexity: we show that a monochromatic rectangle is a *symmetric square*, in certain circumstances.

Generally speaking, our bit complexity results imply that MaxF is harder than Leader, which is harder than Consensus. However, we realized that the complexity of Leader can vary widely depending on the symmetry of the chain and the knowledge of this that the processors may have; which is not the case for Consensus or MaxF. In particular, it is possible to solve Leader, when n is known to be even, sending less than one bit per link in each direction; this is impossible for Consensus.

In what concerns upper bounds in a chain, we assumed that a scheduler eventually always wakes up the terminals, if not waken up by a message. This assumption is essential for the bounds of the kind $c \cdot n \pm O(1)$. In this context, it would be interesting to consider other assumptions on the wake up requirements, for a scheduler.

Our main technical result is an analogue of the result of [5] for communication complexity. We proved that the bit complexity of Leader in an odd length chain with inputs only at the terminals is equal to n times the bit complexity of the problem in a two processor chain. In communication complexity, this kind of property holds for every problem (i.e., function), while in the distributed computing setting, we showed that this is not the case: for example, while $BitC(\texttt{Leader}) = \Theta(\log M)$ in a two processor chain, in an even length chain $BitC(\texttt{Leader}) = \Theta(n)$.

Note that in the ring case, there is a gap between our lower bound of $\Omega(n \log M)$ and the known upper bound of $O(n \log n \log M)$. One can distinguish two different techniques for proving lower bounds for the ring topology: the technique of [4, 2, 10], which exploits the fact that in some executions the ring must be traversed by a long chain of messages, and the technique used here, which shows that many bits must be sent by each processor. It is possible that none of the above two techniques by itself could close the above gap, e.g., by proving an $\Omega(n \log n \log M)$ lower bound on the bit complexity of MaxF in the ring (improving the lower bounds for Leader and Consensus could be even harder). Improving these lower bounds, if at all possible, may require a new technique, which could be a combination of these two techniques.

Acknowledgments

We are grateful to Martin Dietzfelbinger for helpful and enlightening discussions, at the early stages of this research, to Shay Solomon and an anonimous referee for pointing out local gaps in the submitted version of the paper, and to Shay Solomon for his comments on improving the text.

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*, pages 373–382, 1997.

[6] M. Dietzfelbinger, J. Hromkovic, G. Schnitger, "A comparison of two lower-bound methods for communication complexity," *Theoretical Computer Science* **168** (1996), 39-51.

[7] Y. Dinitz, S. Moran and S. Rajsbaum. Bit complexity of breaking and achieving symmetry in paths and rings. In: *Proc. of the 31th Symposium on Theory of Computing, STOC'99*, 265–274.

[8] Y. Dinitz, S. Moran and S. Rajsbaum. Exact Communication Costs for Consensus and Leader in a Tree. *J. of Discrete Algorithms* **1** (2003), 167–183.

[9] Y. Dinitz and N. Solomon. Two Absolute Bounds for Distributed Bit Complexity. *Theoretical Computer Science* **384** (2007), 168-183.

[10] Pavol Duris and Zvi Galil, "Two lower bounds in asynchronous distributed computation," *J. of Computer and System Sciences,* **42**(3), pp. 254–266, June 1991.

[11] Matthias Fitzi and Juan Garay, "Efficient Player-Optimal Protocols for Strong and Differential Consensus." Proc. 22nd ACM Symposium on the Principles of Distributed Computing (PODC'03), pp. 211–220, Boston, MA, July 2003.

[12] 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.

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

[14] Eyal Kushilevitz, Nathan Lineal, Rafail Ostrovsky, "The linear-array conjecture in communication complexity is false," *Combinatorica,* **19** (2), pp. 241–254, 1999.

[15] 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.

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

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

[18] Shlomo Moran, Manfred K. Warmuth, "Gap Theorem in Distributed Computing", *SIAM Journal of Computing* **22:2** (1993), 379–394.

[19] Shlomo Moran, Yaron Wolfsthal, "An extended impossibility result for asynchronous complete networks", *Information Processing Letters*, **26**, pp. 141–151, 1987.

[20] Gerard Tel, *Introduction to Distributed Algorithms,* Cambridge University Press, 1994.

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

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