Electronic Colloquium on Computational Complexity - Reports Series 1996 - available via:

ECCC
TR96-063

FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

The Linear-Array Problem in Communication Complexity Resolved

Martin Dietzfelbinger*
Fachbereich Informatik
Universität Dortmund
D–44221 Dortmund, Germany
email: dietzf@ls2.informatik.uni-dortmund.de

November 6, 1996

Abstract

Tiwari (1987) considered the following scenario: $k+1$ processors $P_0, \ldots, P_k$, connected by k links to form a linear array, are to compute a function $f(x,y)$, $x \in X$, $y \in Y$, on a finite domain $X \times Y$, where the x-part of the input is only known to P_0, the y-part is only known to P_k; the intermediate processors $P_1, \ldots, P_{k-1}$ do not have any information. The processors compute $f(x,y)$ by exchanging binary messages across the links, according to some protocol Φ. Let $D_k(f)$ denote the minimal complexity of such a protocol Φ, i. e., the total number of bits sent across all links for the worst case input, and let $D(f) = D_1(f)$ denote the (standard) 2-party communication complexity of f. Tiwari proved that $D_k(f) \geq k \cdot (D(f) - O(1))$ for almost all functions, and conjectured this inequality to be true for all f. His conjecture was falsified by

*Partially supported by DFG grant Di 412/2-2. This paper appears as Research Report Nr. 632/1996, Fachbereich Informatik, Universität Dortmund, Dortmund, Germany

Kushilevitz, Linial, and Ostrovsky (1996): they exhibited a function f for which $D_k(f)$ is essentially bounded above by $\frac{3}{4}k \cdot D(f)$. The best general lower bound known is $D_k(f) \geq k \cdot (\sqrt{D(f)} - \log k - 3)$. We prove a weakened form of Tiwari's conjecture:

$$D_k(f) \geq \gamma \cdot k \cdot D(f),$$

for arbitrary functions f, where $\gamma = \frac{1}{4}\log_2(3/2) > 0.146$.

Applying the general framework provided by Tiwari, we may derive lower bounds for the communication complexity of computing functions in general asynchronous networks on the basis of the two-party communication complexity of the function: If the network can be decomposed into layers $V_0, \ldots, V_k$, where links are only allowed to connect nodes in the same layer and nodes in adjacent layers, and the input parts x and y are given to nodes in V_0 and V_k, respectively, then the total number of bits that have to be communicated for computing $f(x, y)$ is bounded below by $\Omega(k \cdot D(f)/\log w)$, where w is an upper bound on the number of links connecting any two successive layers.

Moreover, we consider the nondeterministic case. Let $N_k(f)$ denote the nondeterministic communication complexity of f in an array with k links, $N(f)$ the two-party complexity. The best known (trivial) bound is $N_k(f) \geq k \cdot (N(f) - \log k - 2)$, which is useless for functions with very small nondeterministic communication complexity. We prove that $N_k(f) \geq \lceil k/2 \rceil \cdot (N(f) - 2)$. Similarly, the method can also be applied to *public coin* randomized two-party protocols, with consequences for the private coin model.

Our result also implies that lower bounds on the deterministic communication complexity of functions can be used directly to derive corresponding lower bounds for the time complexity of the function on deterministic one-tape Turing machines. The question whether this was possible has been open for a long time.

1 Introduction: Background and Summary

In his fundamental paper on communication complexity of computing functions in networks, Tiwari [20] studied as a basic problem the communication complexity in a linear array of processors. Following Tiwari, we consider a simple network, consisting of $k+1$ processors $P_0, \ldots, P_k$, in which processors

P_{i-1} and P_i are connected by a bidirectional link, for $1 \le i \le k$. These processors are to compute a function $f : X \times Y \to \{0,1\}$ for certain finite sets X, Y.[1] At the beginning of the computation, P_0 receives $x \in X$, P_k receives $y \in Y$, and processors $P_1, \ldots, P_{k-1}$ are in an initial state that does not depend on the input ("know nothing"); at the end of the computation, P_0 and P_k know $f(x,y)$. For the computation, the processors exchange binary messages across the links, following a deterministic protocol Φ fixed beforehand. (Protocols and their properties will be discussed in detail in Section 2. For understanding the remainder of the introduction, these details are not necessary. For a thorough introduction to communication complexity, see [5] or [10].) For measuring performance, the cost of internal computations carried out by each processor is neglected; we are only interested in the cost $L^{\Phi}(x,y)$, which is the total number of bits exchanged on all links taken together in carrying out Φ on input (x,y). The worst-case cost of Φ, denoted by L^{Φ}, is defined as the maximum of all $L^{\Phi}(x,y)$. The complexity of f on an array of length k is defined as

$$D_k(f) = \min\{L^{\Phi} \mid \Phi \text{ is a protocol for } f \text{ on a linear array of length } k\}.$$

We wish to compare $D_k(f)$ with the (standard) two-party communication complexity $D(f) = D_1(f)$. It is clear that $D_k(f) \le k \cdot D(f)$, since any two-party protocol can be run on the linear array: the inner processors only pass on the bits they receive. It is quite natural to conjecture, as Tiwari did, that $D_k(f) \ge k \cdot (D(f) - O(1))$, since there is no obvious way how the inner processors can act as anything but relays. Tiwari showed that his conjecture is true for almost all functions if $|X| = |Y|$, essentially by proving that $D_k(f) \ge k \cdot \log(\mathrm{rank}_{\mathbb{F}}(M_f))$ where M_f is the "communication matrix" $(f(x,y))_{x \in X, y \in Y}$ of f and $\mathbb{F}$ is an arbitrary field.[2] He also showed that "fooling set" arguments for proving lower bounds can be "lifted" from the two-party situation to the array. However, it has been known for a long time (see, e. g., [16]) that $D(f)$ may be much larger than $\log(\mathrm{rank}_{\mathbb{F}}(M_f))$, and that for many functions f fooling set arguments yield bounds very far away from $D(f)$. Thus, Tiwari's methods alone were not sufficient to establish the conjecture.

Recently, Kushilevitz, Linial, and Ostrovsky [9] refuted Tiwari's conjecture by exhibiting a function f for which $D_k(f)$ is (essentially) bounded above by

[1] For notational convenience, we will assume that $X, Y \subseteq \{0,1\}^n$ for some n.

[2] Unless indicated otherwise, all logarithms in this paper are to the base 2.

$\frac{3}{4}k \cdot D(f)$. Their proof of this result is interesting because of two aspects: first, it shows that for functions with a suitable, cleverly chosen structure the presence of intermediate processors does indeed help; second, it introduces an important new technique for establishing lower bounds on $D(f)$. The question what the largest possible distance between $k \cdot D(f)$ and $D_k(f)$ is becomes even more interesting after this result. The best lower bound known until now is

$$D_k(f) \geq k \cdot (\sqrt{D(f)} - \log k - 3)\,,$$

which is obtained as follows [9]: It is easy to transform a deterministic protocol for the array of complexity $D_k(f)$ into a Las Vegas (i. e., zero-error randomized) two-party protocol of complexity at most $\frac{1}{k}D_k(f) + \lceil \log k \rceil$, thus showing that the Las Vegas complexity $R_0(f)$ is bounded above by this term. Then one may apply the well-known relation $D(f) \leq (R_0(f) + 2)^2$ [1, 4, 13].

In this paper, we prove, for arbitrary k and f, that

$$D_k(f) \geq \gamma \cdot k \cdot D(f),$$

where $\gamma = \frac{1}{4}\log(3/2) > 0.146$. As an intermediate step of the proof, we show that

$$D_k(f) \geq \lceil k/2 \rceil \cdot \log(C^P(f)),$$

where $C^P(f)$ is the protocol cover complexity of f, which is the minimum number of different message sequences occurring in any two-party protocol that computes f. With the exception of the new methods from [3] and [6], all existing lower bound proofs for the two-party complexity of a function f work by establishing $C^P(f) \geq b$ for some bound b and then concluding $D(f) \geq \log b$. Combined with the method presented in this paper, each proof of this type entails that $D_k(f) \geq \lceil k/2 \rceil \cdot \log b$, improving in the constant factor the bound achieved by combining the inequalities $D_k(f) \geq \gamma \cdot k \cdot D(f)$ and $D(f) \geq \log b$.

The new proof method can also be used to (less dramatically) improve bounds on the complexity of nondeterministic protocols for the array. Let $N_k(f)$ denote the nondeterministic communication complexity of f on an array of length k, $N(f)$ the two-party complexity. The following lower bound on $N_k(f)$ in terms of $N(f)$ is easily established:

$$N_k(f) \geq k \cdot (N(f) - \log k - 2)\,.$$

This bound is useless for functions with nondeterministic communication complexity smaller than $\log k$. We prove

$$N_k(f) \geq \lceil k/2 \rceil \cdot (N(f) - 2),$$

for arbitrary f and k.

We also consider randomized protocols. Just as for nondeterministic protocols, it is easy to establish certain types of bounds, which cover a wide range of parameters, but do not work for functions with small complexities on long arrays. Our method allows us to fill this gap to some extent. As above, let $R_0(f)$ denote the Las Vegas complexity of f in the two-party model. Further, let $R_\varepsilon(f)$ and $R^1_\varepsilon(f)$ denote the complexity of f in the two-sided error model (error ε, $0 \leq \varepsilon < \frac{1}{2}$, i.e. the randomized protocols Φ admitted satisfy **Prob**(the result of Φ on input (x, y) is $f(x, y)) \geq 1 - \varepsilon$), and in the one-sided error model, respectively. The corresponding complexities for the array of length k are denoted by $R_{0,k}(f)$, $R_{\varepsilon,k}(f)$, $R^1_{\varepsilon,k}(f)$, respectively. It is easy to establish the following bounds:

$$\begin{aligned}
R_{0,k}(f) &\geq k \cdot (R_0(f) - \lceil \log k \rceil); \\
R_{\varepsilon,k}(f) &\geq \delta k \cdot (R_{\varepsilon+\delta}(f) - \lceil \log k \rceil), \text{ for } 0 < \varepsilon < \varepsilon + \delta < \tfrac{1}{2}; \\
R^1_{\varepsilon,k}(f) &\geq \eta k \cdot (R^1_{\varepsilon(1-\eta)+\eta}(f) - \lceil \log k \rceil), \text{ for } 0 < \varepsilon < 1 \text{ and } 0 < \eta < 1.
\end{aligned}$$

By applying standard methods for dealing with probabilistic algorithms, as can be found, e.g., in [12], it is not hard to conclude from the last two inequalities that there is a constant $B_0 \geq 1$ so that

$$\begin{aligned}
R_{\varepsilon,k}(f) &= \Omega(\min\{\tfrac{1}{2} - \varepsilon, 1/\log \varepsilon^{-1}\} \cdot k \cdot (R_\varepsilon(f) - B_0 \log k)), \text{ for } 0 < \varepsilon < \tfrac{1}{2}; \\
R^1_{\varepsilon,k}(f) &= \Omega((1/\log \varepsilon^{-1}) \cdot k \cdot (R^1_\varepsilon(f) - B_0 \log k)), \text{ for } 0 < \varepsilon < 1.
\end{aligned}$$

It is clear that all these bounds are useless for two-party complexities smaller than $\log k$ and that the bounds for two-sided error protocols are weak if the error ε is close to $\frac{1}{2}$. Bounds without these disadvantages can be obtained by a detour via public-coin protocols, which fit together well with our lower bound method. Let $R^{\mathrm{pub}}_{0,k}(f)$, $R^{\mathrm{pub}}_{\varepsilon,k}(f)$, $R^{1,\mathrm{pub}}_{\varepsilon,k}$, $R^{\mathrm{pub}}_0(f)$, $R^{\mathrm{pub}}_\varepsilon(f)$, $R^{1,\mathrm{pub}}_\varepsilon$ be the versions of the randomized complexity bounds for the public-coin model. Using the relationship between the communication complexity with respect to public and private-coin protocols established by Newman [14], we can prove the following (recall that we assume that $X, Y \subseteq \{0,1\}^n$): There are constants $B, B_1, B_2 \geq 1$ such that

$$
\begin{aligned}
R_{0,k}(f) &\geq R_{0,k}^{\text{pub}}(f) = \Omega(k \cdot R_0^{\text{pub}}(f)) = \Omega(k \cdot (R_0(f) - B \cdot \log n));\\
R_{\varepsilon,k}(f) &\geq R_{\varepsilon,k}^{\text{pub}}(f) = \Omega(k \cdot R_\varepsilon^{\text{pub}}(f))\\
&= \Omega(k \cdot (R_\varepsilon(f) - B_1 \log n - B_2 \log(\tfrac{1}{\varepsilon((1/2)-\varepsilon)}))), \text{ for } 0 < \varepsilon < \tfrac{1}{2};\\
R_{\varepsilon,k}^1(f) &\geq R_{\varepsilon,k}^{1,\text{pub}}(f) = \Omega(k \cdot R_\varepsilon^{1,\text{pub}}(f))\\
&= \Omega(k \cdot (R_\varepsilon^1(f) - B_1 \log n - B_2 \log(\tfrac{1}{\varepsilon(1-\varepsilon)}))), \text{ for } 0 < \varepsilon < 1.
\end{aligned}
$$

We must leave it as an open question whether it is possible to obtain bounds for private-coin protocols without incurring a loss by applying Newman's method.

Finally, we turn to applications. First, one should recall that Tiwari [20] showed how bounds on communication complexity for the array can be transformed into bounds on the communication complexity, i. e., the total number of bits transmitted across links, for computing functions on arbitrary asynchronous networks. More specifically, he showed the following. Assume that the network can be split into a sequence of $k+1$ layers numbered $0, \ldots, k$, so that links only run between nodes in the same layer or between nodes in neighboring layers, and that w is an upper bound on the maximal number of links connecting two neighboring layers. If processors in layer 0 know part x of the input, processors in layer k know part y of the input, then computing the value $f(x, y)$ requires that $\Omega(D_k(f)/\log w)$ bits are transmitted overall. (It is quite easy to see that corresponding assertions hold for the nondeterministic and for the randomized case.) Tiwari demonstrated the power of his method by applying it to a sequence of concrete functions for which the rank method or the fooling set method for proving lower bounds could be applied. Our method, together with Tiwari's construction, yields a general lower bound of $\Omega(k \cdot D(f)/\log w)$ for the communication complexity of f in a network as described. Since the reduction from the general network to the array introduces constant factors in the lower bounds anyway, from the point of view of these applications it does not seem to be too important whether the array bound involves a constant factor or not.

As a second application, we show how our method can be applied to transfer communication complexity bounds to time bounds for one-tape Turing machines. Consider the following situation. Let $f : \bigcup_{n\geq 1}(\{0,1\}^n \times \{0,1\}^n) \to \{0,1\}$ be a computable function. The restriction of f to $\{0,1\}^n \times \{0,1\}^n$

is called f_n. Let $time_M(n)$ denote the worst case running time on inputs from $\{0,1\}^n \times \{2\}^{k(n)-1} \times \{0,1\}^n$ of a Turing machine M that computes the function $\hat{f}$, where

$$\hat{f}(x2^{k(n)-1}y) = f(xy), \text{ for arbitrary } x, y \in \{0,1\}^n.$$

(The x- and y-parts of the input are separated by a "bridge" [17] or a "desert" of length $k(n)-1$.) Similarly as in the array case in the randomized setting, one may show by known methods that

$$time_M(n) = \Omega\left(k(n) \cdot \left(\sqrt{D(f_n)} - \min\{\log k(n), B \cdot \log n\}\right)\right),$$

for some constant $B \geq 1$. The obvious question was whether

$$time_M(n) = \Omega\left(k(n) \cdot D(f_n)\right)$$

or not. This statement is an easy consequence of our theorem for the array. Looking a little closer, we may also obtain

$$T(n) = \Omega\left(D(f_n)^2\right),$$

where $T(n)$ is the running time of a Turing machine computing f on inputs from $\{0,1\}^n \times \{0,1\}^n$. Similar results may be obtained for the nondeterministic and randomized situation. Note, however, that in these models for functions with communication complexity at least $(1+\Omega(1))\log k$ bounds can be obtained almost trivially. It should also be mentioned that Paturi and Simon [17] have provided a method that can be used to derive the same bounds for nondeterministic one-tape Turing machines in terms of nondeterministic communication complexity (without making the constants explicit). This method explicitly utilizes the fact that the Turing machine is a uniform computational model and do not carry over to arrays of processors, which are a nonuniform model. In the randomized setting, Kalyanasundaram and Schnitger [7] have introduced a very powerful method for proving lower bounds for randomized one-tape Turing machines with an extra input tape on the basis of communication complexity bounds; these methods, however, only apply to very special functions. The results obtained in [17] on the relationship between randomized communication complexity and randomized one-tape Turing machines refer to the unbounded error model, which is quite different from the more standard bounded-error model used in this paper.

2 Preliminaries: Deterministic protocols and their basic properties

In order to make the paper self-contained, we recall here some basic concepts concerning two-party and array protocols that are important for the later proofs; moreover, we prove some fundamental facts for later use.

It is a slightly delicate issue to specify what a protocol for a linear array should be. Tiwari gave a detailed description of such a concept. In his setting, it was essential for obtaining optimal bounds (taking constant factors into account) that an inner processor in the array was not allowed to "listen" to both its links in the array at the same time. Here, we give a slightly more generous definition, which may also be more natural. Nonetheless, Tiwari's methods apply; thus, all results from [20] are also valid in our model. The specification will serve as a sound basis for our own lower bound proofs.

2.1 Two-party protocols

We first recall how two-party protocols are specified (cf. [10]). There are two players, A(lice) and B(ob). Let $(x, y) \in X \times Y$ be an input. Alice gets the x-part, Bob gets the y-part. The players communicate by exchanging messages consisting of one bit each. The order in which they are allowed to send bits is given by a "protocol tree", which is a rooted binary tree with the following structure: inner nodes are labeled by either "A" or "B" and have two outgoing edges labeled by 0 and 1. After t communication steps the sequence $s = b_1 \cdots b_t$ of bits communicated so far determines a node v in the tree. (Follow the edges labeled $b_1, \ldots, b_t$.) If v is an inner node, its label A or B determines which of Alice and Bob is to send the next bit. If v is a leaf, the communication is finished. It is a trivial observation that the complete message sequences that are possible with such a tree form a prefix-free set of binary strings; each leaf of the tree corresponds to such a message sequence. Moreover, note that from the message sequence the whole communication, i. e., the information which party has sent which bit, can be reconstructed. How do Alice and Bob decide which bit to send? In deterministic protocols, the actions of Alice and Bob are determined by functions

$$\Phi^A : X \times \{0,1\}^* \to \{0,1,\perp\} \text{ and } \Phi^B : \{0,1\}^* \times Y \to \{0,1,\perp\},$$

so that $\Phi^A(x,s) \in \{0,1\}$, i. e., not undefined, if and only if s corresponds to a node in the tree labeled with A; similarly, $\Phi^B(s,y) \in \{0,1\}$ if and only if s corresponds to a node labeled with B. (Since the tree is finite, Φ^A and Φ^B are really finite objects.) In nondeterministic protocols, these functions allow for making nondeterministic choices, i. e.,

$$\Phi^A : X \times \{0,1\}^* \to \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$$

and

$$\Phi^B : \{0,1\}^* \times Y \to \{\emptyset, \{0\}, \{1\}, \{0,1\}\}\,.$$

If s corresponds to an inner node of the protocol tree with label A, then $\Phi^A(x,s) \neq \emptyset$ for all $x \in X$. If $\Phi^A(x,s) = \{b\}$ for $b = 0$ or 1, Alice will send bit b on previous message sequence s and input x; if $\Phi^A(x,s) = \{0,1\}$, she nondeterministically chooses to send 0 or 1. Similar rules apply to Bob. In randomized protocols (the "private coin" version), the functions Φ^A and Φ^B, if defined, associate a probability distribution on $\{0,1\}$ with (x,s) resp. (s,y); technically,

$$\Phi^A : X \times \{0,1\}^* \to [0,1] \cup \{\perp\} \text{ and } \Phi^B : \{0,1\}^* \times Y \to [0,1] \cup \{\perp\}.$$

If s corresponds to a node with label A, then $\Phi^A(x,s) \neq \perp$ for all $x \in X$, and for $p = \Phi^A(x,s)$, Alice will flip a coin that turns up heads ("1") with probability p and tails ("0") with probability $1-p$, and she will send the resulting bit. Similar rules apply to Bob.

All three types of protocols use one fixed tree. It remains to determine what the result of a computation represented by a complete message sequence s is, i. e. one that corresponds to a leaf. We restrict our attention to boolean functions; the possible outcomes are 0 or 1. We adopt the convention that at the end of the computation both parties "know the result", i. e., Alice can compute it from x and s, Bob can compute it from s and y. Using the basic fact that the set of inputs that results in a message sequence s forms a "rectangle" $X' \times Y'$ for some $Y' \subseteq Y$, $X' \subseteq X$, one easily sees that the result bit is a function of s alone. We may thus assume that the leaf of the protocol tree that corresponds to s is labeled with this result bit. This assumption can also be justified in the nondeterministic and the randomized case. The assumption will be made for all protocol trees used in the following.

We say that a deterministic protocol (consisting of a protocol tree and deterministic functions Φ^A and Φ^B) computes $f : X \times Y \to \{0,1\}$ if for all $(x,y) \in X \times Y$ the result of the computation is $f(x,y)$. The complexity of the protocol is the depth of the tree, i. e., the length of the longest possible message sequence. A nondeterministic protocol computes f if for all $(x,y) \in f^{-1}(1)$ there is a computation with result 1, but for $(x,y) \in f^{-1}(0)$ there is none. The complexity of the protocol is

$$\max_{(x,y)\in f^{-1}(1)} \min\{|s| \mid s \text{ is a computation on } (x,y) \text{ with result } 1\},$$

the nondeterministic complexity $N(f)$ of f is the minimum complexity of a protocol that computes f. It is well known that for nonconstant f this is $\lceil \log_2(1 + C_N(f)) \rceil$ or $1 + \lceil \log_2(1 + C_N(f)) \rceil$, where $C_N(f)$ is the minimum number of not necessarily disjoint monochromatic rectangles needed to cover the 1's in the communication matrix M_f.

For randomized protocols, we use the standard notation. The complexity of a protocol that is allowed to make errors is the depth of its tree. Such a protocol computes f with (two-sided) error $0 < \varepsilon < \frac{1}{2}$ if for all $(x,y) \in X \times Y$ we have

$$\mathbf{Prob}(\text{result of protocol on input } (x,y) \text{ equals } f(x,y)) \geq 1 - \varepsilon.$$

The randomized complexity $R_\varepsilon(f)$ is the minimum complexity of a protocol that computes f with error ε. One-sided error protocols are allowed to give wrong answers only for inputs (x,y) with $f(x,y) = 1$; apart from that, the one-sided error randomized complexity $R^1_\varepsilon(f)$ (for $0 < \varepsilon < 1$) is defined in analogy to the two-sided error case. The complexity of a zero-error protocol, which must always compute the correct result, on an input (x,y) is the expected number of bits exchanged on this input.

A different concept is that of a "public-coin" randomized protocol, which really is a probability distribution on a set of deterministic protocols. The corresponding complexity measures R^{pub}_0, $R^{1,\text{pub}}_\varepsilon$, $R^{\text{pub}}_\varepsilon$ are defined as usual. (A rigorous definition for the array is given in Section 5.) Note that public coin protocols are the only type that use different protocol trees for a single function.

2.2 Protocols for the array

In this section, we define protocols for the array of length k, and explore some of their basic properties. Such arrays consist of processors $P_0, P_1, \dots, P_k$, where P_{i-1} and P_i are connected by a bidirectional link, numbered i, for $1 \leq i \leq k$. (Our definition differs slightly, but not essentially, from that one given by Tiwari [20].)

Roughly, a computation for a function f runs as follows. Processor P_0 is given the first part x of the input, processor P_k the second part y. Then the processors send messages to their neighbors and receive messages from them, in a fully asynchronous manner. Without loss of generality, messages are assumed to consist of single bits. In order to decide whether to send a message or not, a processor P_i will look only at the messages exchanged previously with its neighbors P_{i-1} and P_{i+1} (if $1 \leq i < k$) or at its part of the input and the messages exchanged previously with its one neighbor (if $i = 0$ or $i = k$). Processors may put off sending a message and a message may be delayed on a link for an indefinite (but finite) period of time. We require, however, that bits sent across the same link reach their destination in the same order in which they are sent. After a finite number of messages has been exchanged, all processors $P_0, P_1, \dots, P_k$ must (consistently) know the bit that is the result $f(x, y)$ of the computation. (Tiwari demanded that P_0 and P_k know the result and observed that it it easily seen that this entails that all processors know the result. We incorporate this property in the definition. Alternatively, one could consider a different convention, namely that only one processor $P_{i(x,y)}$ knows the result at the end, and that each processor P_i, $i \neq i(x, y)$ knows whether $P_{i(x,y)}$ is to the right or to the left of P_i. The results that can be obtained for this variant are essentially the same.)

We now give a more detailed description of the type of rules used to specify such a protocol. First, consider the link between P_{i-1} and P_i, for some $1 \leq i \leq k$. After some steps, a sequence of bits has been sent back and forth across this link. Both processors decide exclusively on the basis of this history which one of them is next to send a bit across the link. Technically, the communication across the link is governed by a communication tree just as in two-party protocols. Of course, due to delays on the links, the two neighbors may disagree on what the previous sequence of bits is, e. g., if P_i has sent the last bits, it may see a longer sequence of previous messages than P_{i-1}. Note, however, that there will be no crosstalk: at any time, at most one of the processors is allowed to speak, and at points in time when the speaker changes, both views of the previously exchanged messages are the same. As in the two-party situation, the set of possible complete message sequences on the link forms a prefix-free set of binary strings. Also, it is clear that from such a message sequence the whole communication can be reconstructed.

Next, we describe how processors decide which bit to send. In order to avoid tedious case distinctions, it is convenient to regard the inputs x and y as message sequences $s_0 \in \{0,1\}^*$ and $s_{k+1} \in \{0,1\}^*$ on virtual links on which communication is fixed and finished right at the beginning of the computation. For each link i, $1 \leq i \leq k$, there is a function

$$\Phi_i^+ : \{0,1\}^* \times \{0,1\}^* \to \{0,1,\perp\}, (s_{i-1}, s_i) \mapsto \Phi_i^+(s_{i-1}, s_i),$$

which tells P_{i-1} which bit to send to P_i, if any, and a function

$$\Phi_i^- : \{0,1\}^* \times \{0,1\}^* \to \{0,1,\perp\}, (s_i, s_{i+1}) \mapsto \Phi_i^-(s_i, s_{i+1}),$$

which tells P_i which bit to send to P_{i-1}, if any. If, e. g., the previous communication on link i is s_i, and P_{i-1} is next to send a bit to P_i, it may send $\Phi_i^+(s_{i-1}, s_i)$, if this value is different from $\perp$, and nothing (yet) if it equals $\perp$ ("undefined"). The rules for sending bits from right to left are symmetric. There are some simple rules that will ensure that the computation terminates with all communications on links finished and that the computation is uniquely determined in spite of the asynchrony.

(1) (Consistency with protocol trees.) If the node of the protocol tree for link i that corresponds to s_i is labeled with P_i or is a leaf, then $\Phi_i^+(s_{i-1}, s_i)$ is undefined for all s_{i-1}; a symmetric rule applies for $\Phi_i^-(s_i, s_{i+1})$, if the node corresponding to s_i is labeled with P_{i-1}.

(2) (No deliberate blocking.)

(i) Let $1 \le i < k$. If s_i and s_{i+1} are such that P_i is next to speak on both its links, then at least one of $\Phi_i^-(s_i, s_{i+1})$ and $\Phi_{i+1}^+(s_i, s_{i+1})$ is defined.

(ii) Let $0 \le i < k$. If s_i is such that the communication on link i is finished (in particular for $s_i = s_0 = x$), and P_i is next to speak on link $i+1$, then $\Phi_{i+1}^+(s_i, s_{i+1})$ is defined. (A symmetric rule applies for Φ_i^-, $1 \le i \le k$.)

(3) (No reversing decisions.) Let $1 \le i \le k$. If $\Phi_i^+(s_{i-1}, s_i)$ is defined, and $s_{i-1}t_{i-1} = s'_{i-1}$ is an extension of s_{i-1}, then $\Phi_i^+(s'_{i-1}, s_i) = \Phi_i^+(s_{i-1}, s_i)$. (This means that more information received on link $i-1$ or messages sent across link $i-1$ by P_{i-1} will not change the message sent next across link i. A symmetric rule applies for Φ_i^-, $1 \le i \le k$.)

(4) (Unanimity.) After the protocol has finished, the result bits at the leaves of the k protocol trees that are reached all coincide.

In order to illustrate that the rules are sound and make it possible to formulate natural protocols, we recall the obvious way of how to simulate a given two-party protocol for Alice and Bob on the array. Processor P_0 acts exactly as A would, processor P_k as B. The protocol trees for the links are identical to the protocol tree of the two-party protocol. Inner processors just take all bits they receive on their left links and send them out at any time they wish through the right link; they pick up all bits that appear on the right link and send them out through the left link.

No assumptions are made regarding the time at which a processor sends its messages. Indeed, rules (1)–(4) make it possible to completely disregard the notion of time, as is shown by the lemmas to follow. The following lemma, which is central for showing that the concept of complexity of f on an array is well-defined, and will be the technical basis for our proofs, was also stated in Tiwari's paper [20], but no proof was given.

Lemma 2.1 *Let Φ be a deterministic protocol for inputs from $X \times Y$ on an array of length k. Then for each fixed input $(x, y) \in X \times Y$, any execution of Φ will generate exactly the same k message sequences $s_1, \ldots, s_k$ on links $1, \ldots, k$.*

Proof. See Appendix A. ■

The lemma justifies the following definitions.

Definition 2.2 (a) *Let Φ be a protocol for inputs from $X \times Y$ on an array of length k.*

(i) *For input $(x, y) \in X \times Y$, the unique message sequence generated on link i is called $s_i^\Phi(x, y)$ (or $s_i(x, y)$ when Φ can be deduced from the context). The concatenation $s_1^\Phi(x, y) \cdots s_k^\Phi(x, y)$ is denoted by $s^\Phi(x, y)$ or $s(x, y)$.*[3]

(ii) *The cost of Φ is*

$$L^\Phi = \max\{|s^\Phi(x, y)| \mid (x, y) \in X \times Y\},$$

where $|s|$ denotes the length of the binary string s.

(b) *Let $f : X \times Y \to \{0, 1\}$ be a function.*

(i) *A protocol Φ computes f if the result of the computation on (x, y) (i. e., the unique result bit at the leaves corresponding to $s_1(x, y)$, ..., $s_k(x, y)$) is equal to $f(x, y)$.*

(ii) *The complexity of f is $D_k(f) = \min\{L^\Phi \mid \Phi$ is a protocol for the array of length k that computes $f\}$.*

The following very intuitive lemmas are easy consequences of Lemma 2.1. As they are crucial for the further development, we sketch the (mostly straightforward) proofs.

Lemma 2.3 *Let Φ be a deterministic protocol that computes $f : X \times Y \to \{0, 1\}$ on an array of length k. Then for each i, $1 \le i \le k$, there is a two-party protocol Ψ_i for f that uses the protocol tree of Φ for link i and on input (x, y) produces the message sequence $s_i^\Phi(x, y)$, for each $x \in X, y \in Y$.*

[3] Because the possible complete message sequences on the same link form a prefix-free set, k-tuples $(s_1(x, y), \ldots, s_k(x, y))$ and bit sequences $s_1(x, y) \cdots s_k(x, y)$ are in a one-to-one correspondence.

Proof. Let Φ be given. We construct Ψ_i be describing the action taken by Alice and Bob when given inputs x and y respectively. Alice simulates Φ on $P_1, \ldots, P_{i-1}$, Bob simulates Φ on $P_i, \ldots, P_k$. The simulation is done in $2 + |s_i^\Phi(x,y)|$ phases. Assume phase $t-1$ has been completed, Alice and Bob have performed an initial segment of the simulation in their parts, and that $t-1$ bits $s_{i,1}, \ldots, s_{i,t-1}$ have been exchanged. If $t \leq |s_i^\Phi(x,y)|$, one of the two players, Alice, say, is to send the next bit. She advances her simulation of the part of the computation on $P_1, \ldots, P_{i-1}$ in any order until the next bit $s_{i,t}$ is produced that is to cross link i. She sends this bit to Bob, and phase t is finished. If $t-1 = |s_i^\Phi(x,y)|$, Alice and Bob, in two more phases, independently finish their simulations without further communication. By Lemma 2.1, it is inessential in which order in time Alice and Bob run their respective simulations: the two parts taken together yield the unique computation $s_1(x,y), \ldots, s_k(x,y)$; in particular, the communication between Alice and Bob produces $s_i(x,y)$. Since the leaf of the protocol tree is labeled with $f(x,y)$, protocol Ψ_i will also yield this result. ∎

Lemma 2.4 (Cut-and-paste Lemma) *Let Φ be a protocol for an array of length k. If $1 \leq i \leq k$ and $(x,y), (x',y') \in X \times Y$ are such that $s_i^\Phi(x,y) = s_i^\Phi(x',y')$, then*

$$s^\Phi(x,y') = s_1(x,y) \cdots \underbrace{s_i(x,y)}_{=s_i(x',y')} s_{i+1}(x',y') \cdots s_k(x',y').$$

Proof. Choose computations of Φ for (x,y) and (x',y'). We may combine the part of the computation for (x,y) on processors $P_0, \ldots, P_{i-1}$ with the part of the computation for (x',y') on $P_i, \ldots, P_k$ in order to construct a computation for (x,y') on the whole array. The details are similar to the construction in the proof of Lemma 2.3. By Lemma 2.1 the resulting computation produces $s^\Phi(x,y')$. ∎

The following lemma, again intuitively clear, will be crucial for our lower bound argument.

Lemma 2.5 (Bracket Lemma) *Let Φ be a protocol for an array of length k, and let $i \leq j$. Then the message sequences at link i and j determine* *the message sequences at all intermediate links, in the following sense: if* $s_i(x,y) = s_i(x',y')$ *and* $s_j(x,y) = s_j(x',y')$ *for inputs* (x,y), $(x',y') \in X \times Y$, *then* $s_l(x,y) = s_l(x',y')$, *for all* l, $i \leq l \leq k$.

Proof. By Lemma 2.4, applied for link i, we have

$$s(x,y') = s_1(x,y) \cdots s_{i-1}(x,y) s_i(x',y') \cdots s_k(x',y').$$

By applying Lemma 2.4 again, this time to link j and inputs (x,y') and (x,y), we get

$$s(x,y) = s_1(x,y) \cdots s_{i-1}(x,y) s_i(x',y') \cdots s_j(x',y') s_{j+1}(x,y) \cdots s_k(x,y).$$

By the uniqueness property stated in Lemma 2.1 this entails $s_l(x',y') = s_l(x,y)$, for $i \leq l \leq k$. ∎

3 The lower bound for deterministic protocols

In this section, we prove the main result for deterministic protocols. Accordingly, "protocol" means "deterministic protocol" throughout. At the end of the section, we state the general lower bound for arbitrary networks.

Lemma 3.1 (Main Lemma) *Let* Φ *be a protocol that computes* f *on an array of length* k. *Then the following holds.*

(a) *There exists a two-party protocol for* f *that uses no more than* $2^{L^\Phi / \lceil k/2 \rceil}$ *different message sequences.*

(b) *There exists a two-party protocol* Ψ *for* f *of complexity at most*

$$\frac{2 \cdot L^\Phi}{\lceil k/2 \rceil \cdot \log(3/2)}.$$

Proof. (a) Consider link $l = \lceil k/2 \rceil$. According to Lemma 2.3, Φ induces a two-party protocol Ψ_l for f that uses the same protocol tree as Φ on link l. Let c denote the number of leaves of this tree, where it may be assumed that each leaf is reached by some input. Obviously, it is sufficient to show that

$$L^{\Phi} \geq \lceil k/2 \rceil \cdot \log c.$$

For this, fix c inputs $(x_1, y_1), \ldots, (x_c, y_c)$ that lead to different message sequences

$$s_l(x_1, y_1), \ldots, s_l(x_c, y_c)$$

on link l. Assume that k is odd. (The argument for even k is practically the same, with even simpler calculations.) Clearly, we have

$$L^{\Phi} \geq \frac{1}{c} \cdot \sum_{1 \leq r \leq c} |s(x_r, y_r)|.$$

For $1 \leq r \leq c$, we split $s(x_r, y_r) = s_1(x_r, y_r) \cdots s_k(x_r, y_r)$ into $(k-1)/2$ pairs of message sequences at positions i and $k - i + 1$, $1 \leq i < k/2$, so that the central link l is located between these positions, and the message sequence $s_l(x_r, y_r)$ itself:

$$|s(x_r, y_r)| = |s_l(x_r, y_r)| + \sum_{1 \leq i < l} |s_i(x_r, y_r) s_{k-i+1}(x_r, y_r)|,$$

for $1 \leq r \leq c$. This implies

$$L^{\Phi} \geq \frac{1}{c} \cdot \sum_{1 \leq r \leq c} |s_l(x_r, y_r)| + \sum_{1 \leq i < l} \frac{1}{c} \cdot \sum_{1 \leq r \leq c} |s_i(x_r, y_r) s_{k-i+1}(x_r, y_r)|. \tag{3.1}$$

By construction, the sequences $s_l(x_r, y_r)$, $1 \leq r \leq c$, are different; clearly, they form a prefix-free set. In addition, we have that for each i, $1 \leq i < l$, the double sequences

$$s_i(x_r, y_r) s_{k-i+1}(x_r, y_r), \ 1 \leq r \leq c,$$

are different. Indeed, suppose for a contradiction that for some $r \neq t$ we have

$$s_i(x_r, y_r) s_{k-i+1}(x_r, y_r) = s_i(x_t, y_t) s_{k-i+1}(x_t, y_t).$$

Since the message sequences at link i form a prefix-free set, this implies

$$s_i(x_r, y_r) = s_i(x_t, y_t) \text{ and } s_{k-i+1}(x_r, y_r) = s_{k-i+1}(x_t, y_t).$$

Using that $i \leq l \leq k - i + 1$, we conclude from Lemma 2.5 that $s_l(x_r, y_r) = s_l(x_t, y_t)$, which is the desired contradiction.

It is also easy to see, using the fact that both the message sequences at link i and at link $k-i+1$ form prefix-free sets, that the set $\{s_i(x_r, y_r)s_{k-i+1}(x_r, y_r) \mid 1 \leq r \leq c\}$ is prefix-free. Thus, we may apply the following well-known fact. (It was used in [20] in a similar way.)

Fact 3.2 *If $t_1, \ldots, t_c$ are strings in $\{0,1\}^*$, and none of the t_r is a proper prefix of another, then*

$$\frac{1}{c} \cdot \sum_{1 \leq r \leq c} |t_r| \geq \log c.$$

If we apply this fact to each of the l average values in equation (3.1), we obtain that the right hand side is bounded below by $\log c + \sum_{1 \leq i < l} \log c = \lceil k/2 \rceil \cdot \log c$; thus,

$$L^\Phi \geq \lceil k/2 \rceil \cdot \log c,$$

as desired.

(b) To derive (b) from (a), we apply a folklore fact that relates the "protocol cover complexity" and the deterministic communication complexity of a function f [10]. The fact and the proof are closely related to a similar relation between formula size and depth that has been proved by Spira [19]. It seems that a proof has not been published before, excepting in the forthcoming book [10]; so for the convenience of the reader, we supply one in an Appendix.

Fact 3.3 *If f has a two-party protocol with at most $c \geq 1$ different message sequences (i.e., the protocol tree has c leaves), then f has a two-party protocol of complexity at most $2 \log_{3/2} c$.*

Proof. See Appendix B. ∎

Part (a) established the upper bound $\log c \leq L^\Phi / \lceil k/2 \rceil$ for the number of leaves in the protocol tree for Ψ_l. Applying Fact 3.2 yields another protocol

Ψ for f of complexity bounded by

$$2\log_{3/2} c \leq 2\log(3/2)^{-1}\log c \leq \frac{2L^{\Phi}}{\lceil k/2\rceil \cdot \log(3/2)},$$

as claimed. ∎

Theorem 3.4 *For arbitrary $f : X \times Y \to \{0,1\}$ and $k \geq 2$ we have*

$$D_k(f) \geq \frac{1}{2}\log(3/2) \cdot \lceil k/2\rceil \cdot D(f),$$

which entails

$$D_k(f) > 0.146 \cdot k \cdot D(f).$$

Proof. Immediate from part (b) of the Main Lemma. ∎

Remark 3.5 A brief comment on the methods employed in the proof of the Main Lemma may be in order. Already in Tiwari's paper the approach via an average over certain inputs and the use of Fact 3.2 was taken in all lower bound arguments for the array, including the proof of the rank lower bound. The essential new part of the present proof is the use of the Bracket Lemma 2.5.

Remark 3.6 Excepting the arguments from [6, 9], all known lower bound proofs for the deterministic communication complexity $D(f)$ of a function f work in two steps: show that every protocol tree for f has at least b leaves and conclude $D(f) \geq \log b$. By part (a) of the Main Lemma, any such proof will entail that $D_k(f) \geq \lceil k/2\rceil \cdot \log b$, which will be sharper than the bound given by combining the inequality $D_k(f) > 0.146k \cdot D(f)$ from the Main Theorem with the bound $D(f) \geq \log b$.

We finish our discussion of the deterministic case with a formulation of the consequence the Main Theorem has for computing in arbitrary asynchronous networks.

Theorem 3.7 *Assume G is a network with bidirectional links whose node set can be split into $k+1$ layers $V_0, V_1, \ldots, V_k$, so that links only connect nodes within one layer V_i or nodes in V_{i-1} with nodes in V_i for some $1 \leq i \leq k$. Let w be the maximal number of links connecting nodes from V_{i-1} with nodes from V_i, for $1 \leq i \leq k$. Assume that G asynchronously computes a function $f : X \times Y \to \{0,1\}$ in such a way that input part x is made known to nodes in V_0 and input part y is made known to nodes in V_k, and that the processors of the other layers start in some fixed initial state. Then the total number of bits that are sent through edges in the computation on the worst case input is bounded below by*

$$\Omega\left(\frac{k \cdot D(f)}{1 + \log w}\right).$$

Proof. This follows by combining Theorem 3.4 with the proof of Theorem 12 in [20]. ∎

4 The lower bound for nondeterministic protocols

Before stating results, we must briefly describe what a nondeterministic protocol for computing f on an array is, and how we measure the cost of such a protocol. Exactly as in the deterministic case (cf. Section 2.2), communication on each link i is governed by a protocol tree for processors P_{i-1} and P_i, whose leaves also give the result (0 means "reject", 1 means "accept") of the communication. Only the way the processors decide which message to send changes. The value of the mappings $\Phi_i^-(s_i, s_{i+1})$, $\Phi_i^+(s_{i-1}, s_i)$ may now be $\emptyset$ (which corresponds to $\perp$ and means that nothing is sent now), $\{0\}$ or $\{1\}$ (send this bit), or $\{0,1\}$ (send either bit, nondeterministically). Rules (1), (2) (no deliberate blocking), and (3) (no reversing decisions) apply without changes. Thus, every legal computation reaches leaves in all k protocol trees. Of course, the outcome of a computation now depends on the nondeterministic choices of the processors. However, if these are fixed (for $\Phi_i^-(s_i, s_{i+1}) = \{0,1\}$, pick one the the two values beforehand, similarly for the Φ^+-functions), the resulting communication on the links is fixed, which means that Lemma 2.1 applies with minor modifications.

Exactly as for the deterministic protocols, we demand that at the end of any legal computation all processors agree on the result, i. e., in all k protocol trees an accepting leaf has been reached or in all protocol trees a rejecting leaf has been reached.

Definition 4.1 *Let Φ be a nondeterministic protocol for inputs from $X \times Y$.*

(a) *We say that Φ accepts (x, y) if there is a computation of Φ on (x, y) in which on all links an accepting leaf is reached (an accepting computation). Otherwise Φ rejects (x, y) (all computations end at a rejecting leaf on all links).*

(b) *Φ computes $f : X \times Y \to \{0, 1\}$ if Φ accepts exactly those $(x, y) \in X \times Y$ with $f(x, y) = 1$.*

We need the counterparts of Lemmas 2.1 through 2.5 from Section 3.

Lemma 4.2 *Let Φ be a nondeterministic protocol on an array of length k. If the nondeterministic choices of the Φ_i^-- and Φ_i^+-functions are fixed in an arbitrary way, then the message sequences that appear at the links on an input (x, y) are the same for every way of running the protocol.*

Proof. Fixing the nondeterministic choices renders a deterministic protocol. Thus Lemma 2.1 applies. ■

We now define the cost of a nondeterministic protocol. For this, we only count accepting computations and among those only the cheapest one. This is justified since we are proving lower bounds on the complexity in the array. Note that it is not obvious (in contrast to the two-party case) how one could cut down the cost of *all* computations to the cost of the cheapest computation on the worst case input $(x, y) \in f^{-1}(1)$.

Definition 4.3 (a) *Let Φ be a nondeterministic protocol that computes f on an array of length k. The complexity of Φ on an input $(x,y) \in f^{-1}(1)$ is*

$$L^\Phi(x,y) = \min\{|s(x,y)| \mid s(x,y) = s_1(x,y)s_2(x,y)\cdots s_k(x,y) \text{ is an accepting computation of } \Phi \text{ on input } (x,y)\}.$$

The complexity of Φ, denoted by L^Φ, is the maximum of the values $L^\Phi(x,y)$, taken over all inputs $(x,y) \in f^{-1}(1)$.

(b) *The nondeterministic complexity of f on the array is*

$$N_k(f) = \min\left\{L^\Phi \mid \Phi \text{ is a nondeterministic protocol that computes } f\right\}.$$

Lemma 4.4 (Lemma 2.3, nondeterministic version) *Let Φ be a nondeterministic protocol for the array of length k, and let $1 \le i \le k$ be arbitrary. Then there is a nondeterministic two-party protocol Ψ_i on the basis of the same protocol tree that is used by Φ on link i, so that a message sequence s_i is legal for (x,y) in Ψ_i if and only if there is a computation $s = s_1 s_2 \cdots s_i \cdots s_k$ for (x,y) in Φ.*

Proof. Practically the same as in the deterministic case. ■

Lemma 4.5 (Nondeterministic Cut-and-paste Lemma) *Let Φ be a nondeterministic protocol for the array of length k, and let $1 \le i \le k$. If $s = s_1 \cdots s_k$ is an accepting computation for (x,y) and $s' = s'_1 \cdots s'_k$ is an accepting computation for (x',y'), and if $s_i = s'_i$, then the mixed sequence $s_1 \cdots s_{i-1} s'_i s'_{i+1} \cdots s'_k$ is an accepting computation for input (x,y').*

Proof. As in the deterministic case. ■

As a consequence, we can note obvious bounds on $N_k(f)$. Let $N(f) = N_1(f)$ be the two-party nondeterministic communication complexity of f.

Observation 4.6 (a) $N_k(f) \le k \cdot N(f)$.

(b) $N_k(f) \geq k \cdot (N(f) - \log k - 2)$.

Proof. (a) Executing an arbitrary two-party protocol with $P_1, \ldots, P_{k-1}$ as relays will yield a protocol for the array, cf. Section 2.2.
(b) Let Φ be a nondeterministic protocol for the array of complexity $\Phi = N_k(f)$. We describe a two-party protocol for players Alice and Bob. On input (x, y), Alice nondeterministically chooses a link i and an accepting message sequence s_i for link i of length at most $\frac{1}{k}N_k(f)$. She checks whether Φ admits an (accepting) computation on processors $P_0, \ldots, P_{i-1}$, with respect to her input part x and some $y' \in Y$, that produces the message sequence s_i on link i. If not, she sends a code for the number 0. If so, she sends a code for the number i and the message sequence s_i to Bob, who checks whether Φ on $P_i, \ldots, P_k$ with input (x', y) for some $x' \in X$ also admits an accepting computation that fits together with s_i. If so, Bob sends back "1", otherwise "0". Clearly, the total communication is at most $\lceil \log(k+1) \rceil + \frac{1}{k}N_k(f) + 1$. The correctness of the protocol follows from the Cut-and-paste Lemma 4.5. Thus, $N(f) \leq \lceil \log(k+1) \rceil + \frac{1}{k}N_k(f) + 1 = \lfloor \log k \rfloor + \frac{1}{k}N_k(f) + 2$, which implies the claim. ∎

In many situations, $N(f)$ will be significantly larger than $\log k$, so that part (b) of the observation may be applied. For large k and small $N(f)$, however, Observation 4.6(b) is useless. Our aim in the remainder of this section is to show that $N_k(f) = \Omega(k \cdot (N(f) - 2))$ regardless of the relation between $N(f)$ and $\log k$.

Due to nondeterministic choices, a counterpart of Lemma 2.5 does not hold: There may be accepting computations $s_1 \cdots s_k$ for (x, y), $s'_1 \cdots s'_k$ for (x', y') with $s_i = s'_i$ and $s_j = s'_j$, $1 \leq i < l < j \leq k$, but still $s_l \neq s'_l$. However, this problem can be overcome by selecting a unique accepting computation for each $(x, y) \in f^{-1}(1)$.

Definition 4.7 *Assume protocol Φ computes f. For $(x, y) \in f^{-1}(1)$, let*

$$s(x, y) = s^{\Phi}(x, y) = s_1(x, y) \cdots s_k(x, y)$$

be the unique accepting computation $s_1 \cdots s_k$ on input (x, y), so that for all other computations $s'_1 \cdots s'_k$ either

$$|s_1 \cdots s_k| < |s'_1 \cdots s'_k|$$

or

$$|s_1 \cdots s_k| = |s'_1 \cdots s'_k|$$

and $s_1 \cdots s_k \leq s'_1 \cdots s'_k$ *in the natural order of equal-length bitstrings.*

Lemma 4.8 (Nondeterministic Bracket Lemma) *Let* Φ *be a nondeterministic protocol, and let* $s(x,y)$*, for* $(x,y) \in f^{-1}(1)$*, be defined as above. Then, for* $(x,y),(x',y') \in f^{-1}(1)$ *and* $1 \leq i \leq l \leq j \leq k$*, we have that*

$$s_i(x,y) = s_i(x',y') \text{ and } s_j(x,y) = s_j(x',y')$$

implies

$$s_l(x,y) = s_l(x',y').$$

Proof. Let $s(x,y) = s_1 \cdots s_k$ and $s(x',y') = s'_1 \cdots s'_k$. As in the proof of the deterministic version (using the Cut-and-paste Lemma twice) we see that

$$\hat{s} := s_1 \cdots s_{i-1} s'_i \cdots s'_j s_{j+1} \cdots s_k$$

is an accepting computation for (x,y) and

$$\hat{s}' := s'_1 \cdots s'_{i-1} s_i \cdots s_j s'_{j+1} \cdots s'_k$$

is an accepting computation for (x',y'). Since $s(x,y)$ and $s(x',y')$ are computations of minimal length for their respective inputs, we immediately get that

$$|s'_i \cdots s'_j| = |s_i \cdots s_j|,$$

hence that $\hat{s}$ is also a minimal length computation for (x,y) and $\hat{s}'$ is one for (x',y'). Thus,

$$s \leq \hat{s} \text{ and } s' \leq \hat{s}'$$

in the natural order of bitstrings of the same length. Clearly, this implies that $s'_i \cdots s'_j = s_i \cdots s_j$. From this we conclude that $s_l = s'_l$ for all l, $i \leq l \leq j$, because the possible message sequences on the same link form a prefix-free set. ∎

Theorem 4.9 *For any* $k \geq 2$ *and any* $f : X \times Y \to \{0,1\}$ *we have*

$$N_k(f) \geq \lceil k/2 \rceil \cdot (N(f) - 2).$$

Proof. (We focus on the changes to be made in comparison to the proof of Theorem 3.4.) Fix $k \geq 2$ and consider an optimal nondeterministic protocol Φ for f on an array of length k. I.e., we have $L^\Phi = N_k(f)$. For $l := \lfloor k/2 \rfloor$ define $c := |\{s_l(x,y) \mid (x,y) \in f^{-1}(1)\}|$, for the unique accepting computations $s(x,y) = s_1(x,y)\cdots s_k(x,y)$ chosen as before. Choose c inputs $(x_1,y_1),\ldots,(x_c,y_c) \in f^{-1}(1)$ such that $s_l(x_1,y_1),\ldots,s_l(x_c,y_c)$ are different. Assume that k is odd. As in the deterministic case, we have

$$L^\Phi \geq \frac{1}{c} \cdot \sum_{1 \leq r \leq c} |s_l(x_r,y_r)| + \sum_{1 \leq i < l} \frac{1}{c} \cdot \sum_{1 \leq r \leq c} |s_i(x_r,y_r)s_{k-1+1}(x_r,y_r)|\,,$$

and may conclude, using Lemmas 4.4 and 4.5 as well as Fact 3.2, that

$$\frac{1}{c} \cdot \sum_{1 \leq r \leq c} |s(x_r,y_r)| \geq \lceil k/2 \rceil \cdot \log c,$$

whence we get

$$N_k(f) \geq \lceil k/2 \rceil \cdot \log c. \tag{4.2}$$

In Lemma 4.4 we noted that Φ induces a two-party protocol Ψ_l on link l. But this protocol is useless here, since it may use more than c message sequences. Instead, we define another, possibly cheaper protocol as follows: On input (x,y), Alice (seeing x) nondeterministically guesses some y' such that $f(x,y') = 1$. (If no such y' exists, Alice sends a default message.) Protocol Φ determines $s(x,y') = s_1(x,y')\cdots s_l(x,y')\cdots s_k(x,y')$. Alice sends $s_l(x,y')$ to Bob. Bob (seeing y) checks whether there is some x' such that $s_l(x',y)$ is identical to $s_l(x,y')$. If so, he sends 1 ("accept"), else 0 ("reject"), which will be the result of this computation. It is easily checked (using the Cut-and-paste Lemma) that this protocol computes f, and that Alice uses at most $c+1$ different messages. Thus, $N(f) \leq \lceil \log(c+1) \rceil + 1$, which entails

$$\log c \geq N(f) - 2.$$

Plugging this into inequality (4.2), we get

$$N_k(f) \geq \lceil k/2 \rceil \cdot (N(f) - 2)\,,$$

as claimed. ∎

5 Randomized protocols: Public coins

An analogue of Theorems 3.4 and 4.9 can also be proved for randomized protocols on the array and for randomized two-party communication complexity. There are several different ways of defining randomized communication complexity, notably, the public-coin model and the private-coin model. For a thorough discussion of these models and their relationship see [14]. We first deal with public-coin protocols, which involve a global random experiment the result of which all processors are told for free. It will turn out that the proof method from previous sections works particularly nicely in combination with public-coin protocols.

Definition 5.1 (a) *A public-coin randomized protocol Φ for X, Y is a family $(\Phi_\alpha)_{\alpha\in I}$ of deterministic protocols for inputs from $X \times Y$ together with a probability distribution, i. e., a family $(p_\alpha)_{\alpha\in I}$ with $p_\alpha \geq 0$ and $\sum_{\alpha\in I} p_\alpha = 1$. We also write $(\Phi_\alpha, p_\alpha)_{\alpha\in I}$ for such a protocol.*

(b) *If $f : X \times Y \to \{0,1\}$ and Φ is a randomized protocol, then we say that*

(i) *Φ computes f with zero error (or: Φ is a Las Vegas protocol) if for all $(x,y) \in X$ we have that $\mathbf{Prob}(\Phi_\alpha$ on input (x,y) yields $f(x,y)) = 1$. The cost of Φ on (x,y) is $L^\Phi(x,y) := \mathbf{E}(L^{\Phi_\alpha}(x,y))$, the cost of Φ is $L^\Phi = \max\{L^\Phi(x,y) \mid (x,y) \in X \times Y\}$.*

(ii) *For $0 < \varepsilon < \frac{1}{2}$, Φ computes f with (two-sided) error bounded by ε if for all $(x,y) \in X$ we have that $\mathbf{Prob}(\Phi_\alpha$ on input (x,y) yields $f(x,y)) \geq 1 - \varepsilon$. The cost of Φ on (x,y) is $L^\Phi(x,y) = \max\{L^{\Phi_\alpha}(x,y) \mid \alpha \in I\}$, the cost of Φ is $\max\{L^\Phi(x,y) \mid (x,y) \in X \times Y\}$.*

(iii) *For $0 < \varepsilon < 1$, Φ computes f with one-sided error bounded by ε if $\mathbf{Prob}(\Phi_\alpha$ on input (x,y) yields $0) = 1$ for all $(x,y) \in f^{-1}(0)$ and $\mathbf{Prob}(\Phi_\alpha$ on input (x,y) yields $1) \geq 1 - \varepsilon$ for all $(x,y) \in f^{-1}(1)$. The cost measures are the same as in* (ii).

(c) *The complexities of f on the array of length k with respect to these measures are defined as follows:*

(i) $R^{\text{pub}}_{0,k}(f) = \inf\{L^\Phi \mid \Phi$ *is a zero-error protocol that computes* $f\}$.

(ii) $R^{\text{pub}}_{\varepsilon,k}(f) = \min\{L^\Phi \mid \Phi$ *is a protocol for* f *with 2-sided error* $\varepsilon\}$, *for* $0 < \varepsilon < \frac{1}{2}$.

(iii) $R^{1,\text{pub}}_{\varepsilon,k}(f) = \min\{L^\Phi \mid \Phi$ *is a protocol for* f *with 1-sided error* $\varepsilon\}$, *for* $0 < \varepsilon < 1$.

The corresponding notions for two-party communication are obtained by choosing $k = 1$. *The two-party complexities are denoted by* R^{pub}_0, $R^{\text{pub}}_\varepsilon$, $R^{1,\text{pub}}_\varepsilon$.

It is very easy to establish connections between randomized complexities of f on the array and in the two-party model. One direction is trivial, as usual:

Observation 5.2 *For all* f *we have the following.*

(a) $R^{\text{pub}}_{0,k}(f) \leq k \cdot R^{\text{pub}}_0(f)$;

(b) $R^{\text{pub}}_{\varepsilon,k}(f) \leq k \cdot R^{\text{pub}}_\varepsilon(f)$, *for* $0 < \varepsilon < \frac{1}{2}$,

(c) $R^{1,\text{pub}}_{\varepsilon,k}(f) \leq k \cdot R^{1,\text{pub}}_\varepsilon(f)$, *for* $0 < \varepsilon < 1$. ■

There are versions for the reverse directions that also have known (and straightforward) proofs. In the case of Las Vegas protocols the result is optimal; in the other two cases constant factors are involved that depend on the error bound.

Proposition 5.3 *For any* f *we have*

(a) $R^{\text{pub}}_{0,k}(f) \geq k \cdot R^{\text{pub}}_0(f)$;

(b) $R^{\text{pub}}_{\varepsilon,k} \geq \delta k \cdot R^{\text{pub}}_{\varepsilon+\delta}(f)$, *for* $0 < \varepsilon < \varepsilon + \delta < \frac{1}{2}$;

(c) $R^{\text{pub}}_{\varepsilon,k}(f) = \Omega(\min\{\frac{1}{2} - \varepsilon, 1/\log \varepsilon^{-1}\} \cdot k \cdot R^{\text{pub}}_\varepsilon(f))$, *for* $0 < \varepsilon < \frac{1}{2}$;

(d) $R^{1,\text{pub}}_{\varepsilon,k}(f) \geq \eta k \cdot R^{1,\text{pub}}_{\varepsilon(1-\eta)+\eta}(f)$, *for* $0 < \varepsilon < 1$ *and* $0 < \eta < 1$;

(e) $R^{1,\mathrm{pub}}_{\varepsilon,k}(f) = \Omega((1/\log\varepsilon^{-1}) \cdot k \cdot R^{\mathrm{pub}}_{\varepsilon}(f))$, *for* $0 < \varepsilon < 1$.

Proof. (a) Let $\Phi = (\Phi_\alpha, p_\alpha)_{\alpha\in I}$ be an arbitrary zero-error protocol for f. Consider the two-party protocol that works as follows. On input (x,y), Alice and Bob choose a link i, $1 \le i \le k$, uniformly at random by a public coin experiment, and then choose some Φ_α according to the distribution $(p_\alpha)_{\alpha\in I}$. Then they run the two-party protocol $\Phi_{\alpha,i}$ that is induced on link i by Φ_α (see Lemma 2.3). It is easily verified that this randomized two-party protocol always yields the correct result and that its complexity is bounded above by $\frac{1}{k}\cdot L^\Phi$. This implies $R^{\mathrm{pub}}_0(f) \le \frac{1}{k}\cdot R^{\mathrm{pub}}_{0,k}(f)$, as desired.

(b) The approach is similar as in (a). We start with a protocol Φ that has error ε and complexity $L^\Phi = R^{\mathrm{pub}}_{\varepsilon,k}(f)$. On input (x,y), a link i and a protocol Φ_α is chosen. Alice and Bob run $\Phi_{\alpha,i}$ on (x,y). If $\Phi_{\alpha,i}$ makes more than $L^\Phi/\delta k$ steps on input (x,y), they break off their communication and settle for the result 0. By the assumption, this is the case for at most a fraction of δ of the links. Thus, the complexity of the new two-party protocol is at most $L^\Phi/\delta k$, and the error is bounded by $\varepsilon+\delta$. Thus,

$$R^{\mathrm{pub}}_{\varepsilon+\delta}(f) \le (1/\delta k)\cdot R^{\mathrm{pub}}_{\varepsilon,k},$$

as claimed.

(c) This claim is derived from (b) by standard methods. If $\frac{1}{4} \le \varepsilon < \frac{1}{2}$, one applies (b) for $\delta = \frac{1}{2}\left(\frac{1}{2}-\varepsilon\right)$ and uses the fact that $R^{\mathrm{pub}}_\varepsilon(f) = O(R^{\mathrm{pub}}_{(\varepsilon+1/2)/2}(f))$, for $\frac{1}{4} \le \varepsilon < \frac{1}{2}$. If $0 < \varepsilon < \frac{1}{4}$, one applies (b) for $\delta = \frac{1}{8}$ and uses the fact that $R^{\mathrm{pub}}_\varepsilon(f) = (R^{\mathrm{pub}}_{3/8}(f)/\log\varepsilon^{-1})$, for $0 < \varepsilon < \frac{1}{2}$.

(d) The proof is similar as that in (b). However, Alice and Bob break off the computation if more than $L^\Phi/\eta k$ steps have been made. By independence, the total error probability of the new two-party protocol is bounded by $\varepsilon + (1-\varepsilon)\eta = \varepsilon(1-\eta)+\eta$.

Part (e) is proved in analogy to (c). ■

The result for Las Vegas protocols is optimal. However, for very large (i. e., close to $\frac{1}{2}$) and very small error bounds the inequalities are weak. We next show that there are bounds for $R^{\mathrm{pub}}_{\varepsilon,k}$ and $R^{1,\mathrm{pub}}_{\varepsilon,k}$ that do not involve extra factors that depend on ε.

Theorem 5.4 *For any f we have:*

(a) $R^{\mathrm{pub}}_{\varepsilon,k}(f) \geq \frac{1}{4}\log(3/2)\cdot k\cdot R^{\mathrm{pub}}_{\varepsilon}(f)$, *for any ε, $0<\varepsilon<\frac{1}{2}$.*

(b) $R^{1,\mathrm{pub}}_{\varepsilon,k}(f) \geq \frac{1}{4}\log(3/2)\cdot k\cdot R^{1,\mathrm{pub}}_{\varepsilon}(f)$, *for any ε, $0<\varepsilon<1$.*

Proof. We only prove (a), since (b) may be treated similarly. Fix a randomized protocol $\Phi=(\Phi_\alpha,p_\alpha)_{\alpha\in I}$ for f with error bound ε and worst case cost $R^{\mathrm{pub}}_{\varepsilon,k}(f)$. We construct a public-coin two-party protocol as follows. The observation made at the end of the proof of the Main Lemma 3.1 can be used to see that from each Φ_α we may obtain a deterministic two-party protocol Ψ_α that computes the same function as Φ_α and satisfies

$$L^{\Psi_\alpha}\cdot\lceil k/2\rceil\cdot\frac{1}{2}\cdot\log(3/2)\leq L^{\Phi_\alpha}\leq L^{\Phi}.$$

The family

$$\Psi=(\Psi_\alpha,p_\alpha)_{\alpha\in I}$$

clearly forms a two-party protocol for f with the same error bound as Φ; the cost L^Ψ of Ψ satisfies $R^{\mathrm{pub}}_{\varepsilon}(f)\leq L^\Psi$, and $L^\Psi\cdot\lceil k/2\rceil\cdot\frac{1}{2}\cdot\log(3/2)\leq L^\Phi$, as desired. ∎

Remark 5.5 There is a very important method for proving lower bounds on two-party probabilistic communication complexity which yields lower bounds on $R^{\mathrm{pub}}_{\varepsilon}(f)$, namely the approach via distributional communication complexity. Given $0<\varepsilon<\frac{1}{2}$ and a measure μ on $X\times Y$, we say a deterministic protocol Ψ computes f with distributional error ε if $\mu(\{(x,y)\mid \Psi$ on (x,y) yields the wrong result$\})\leq\varepsilon$. $D_{\mu,\varepsilon}(f)$ denotes the minimal complexity of a deterministic protocol that computes f with distributional error ε w. r. t. μ. It is well known ([21]) that

$$R^{\mathrm{pub}}_{\varepsilon}(f)\geq D_{\mu,\varepsilon}(f),$$

for all measures μ and $0 < \varepsilon < \frac{1}{2}$. Powerful methods are known for proving lower bounds on $D_{\mu,\varepsilon}(f)$ for important functions f, and for carefully chosen μ. (See, e. g., [2], [18].) Our method shows that $R^{\mathrm{pub}}_{\varepsilon,k}(f) \geq 0.146 \cdot k \cdot D_{\mu,\varepsilon}(f)$ for all measures μ and all $0 < \varepsilon < \frac{1}{2}$. (Similar remarks apply for $R^{1,\mathrm{pub}}_{\varepsilon,k}$.) Note, however, that there are other methods for proving lower bounds on probabilistic communication complexity (in the private-coin version) besides the one just sketched, see e. g. [8], which are not so easily combined with our method.

6 Randomized protocols: Private coins

The usual notion of randomized communication complexity is not that based on the public-coin model as discussed in the previous section, but rather the "private-coin" model, in which the parties carry out their own random experiments, and communicating random bits is not free. (For a thorough discussion of these models and their relationship see [14].) We can derive results on the relationship between complexities on the linear array and in the two-party case model in the private-coin model by a detour via public-coin protocols. We start with defining private-coin randomized protocols for the array. This notion is obtained by generalizing deterministic protocols in the same way as for the two-party model (cf. Section 2.2). Communication on each link is governed by one protocol tree, as in Section 2.2. Only the range of the functions Φ_i^+ and Φ_i^- is changed:

$$\Phi_i^+ : \{0,1\}^* \times \{0,1\}^* \to [0,1] \cup \{\perp\},$$

similarly for Φ_i^-. Assume the communication on link i (link $i+1$) has lead to a node that belongs to the partial message sequence s_i (s_{i+1}). If $\Phi_i^+(s_i, s_{i+1}) = \perp$, processor P_i will not send anything now across link $i+1$. If $\Phi_i^+(s_i, s_{i+1}) = p \in [0,1]$, processor P_i performs a random experiment that has outcome 1 with probability p and outcome 0 with probability $1-p$, and sends the resulting bit to P_{i+1}. All other rules formulated for deterministic protocols remain unchanged. Of course, given an input (x,y) and a randomized protocol Φ, the probability that Φ yields value 1 (resp. 0) is well-defined. It is very useful, at times, to assume that all possible random experiments are carried out beforehand, so that a deterministic protocol results that can be applied to the input. In this way, it is also clear that private-coin protocols are a special case of public-coin protocols. Thus, the following definition is legitimate, cf. Definition 5.1.

Definition 6.1 *Let $f : X \times Y \to \{0,1\}$.*

(a) *The zero-error complexity of f w. r. t. private-coin protocols, denoted by $R_{0,k}(f)$, is defined as $\inf\{L^\Phi \mid \Phi$ is a private-coin zero-error protocol for the array of length k that computes $f\}$.*

(b) *The two-sided error complexity $R_{\varepsilon,k}(f)$ of f w. r. t. protocols, for $0 < \varepsilon < \frac{1}{2}$, and the one-sided error complexity $R^1_{\varepsilon,k}(f)$, for $0 < \varepsilon < 1$, are defined accordingly.*

(c) *The corresponding complexities for the two-party model are denoted by $R_0(f)$, $R_\varepsilon(f)$, $R^1_\varepsilon(f)$, respectively.*

Observation 6.2 *For any f we have*

(a) $R_{0,k}(f) \geq R^{\mathrm{pub}}_{0,k}(f) \geq k \cdot R^{\mathrm{pub}}_0(f)$;

(b) $R_{\varepsilon,k}(f) \geq R^{\mathrm{pub}}_{\varepsilon,k}(f) \geq \frac{1}{4}\log(3/2) \cdot k \cdot R^{\mathrm{pub}}_\varepsilon(f)$, *for* $0 < \varepsilon < \frac{1}{2}$;

(c) $R^1_{\varepsilon,k}(f) \geq R^{1,\mathrm{pub}}_{\varepsilon,k}(f) \geq \frac{1}{4}\log(3/2) \cdot k \cdot R^{1,\mathrm{pub}}_\varepsilon(f)$, *for* $0 < \varepsilon < 1$.

Proof. In each case, the first inequality if trivial; the second inequalities are given by Proposition 5.3(a) and Theorem 5.4(a)(b), respectively. ∎

However, in this section we ask for bounds on $R_{0,k}(f)$, $R_{\varepsilon,k}(f)$, and $R^1_{\varepsilon,k}(f)$ that involve the corresponding *private-coin* two-party complexities. It is clear that the upper bounds from Observation 5.2 also hold in the private-coin model. If we prove bounds similarly as in Proposition 5.3, a slight change is necessary, which weakens the bounds in a typical way.

Proposition 6.3 *For any f we have*

(a) $R_{0,k}(f) \geq k \cdot (R_0(f) - \lceil \log k \rceil)$;

(b) $R_{\varepsilon,k}(f) \geq \delta k \cdot (R_{\varepsilon+\delta}(f) - \lceil \log k \rceil)$, *for* $0 < \varepsilon < \varepsilon + \delta < \frac{1}{2}$;

(c) $R^1_{\varepsilon,k}(f) \geq \eta k \cdot (R^1_{\varepsilon(1-\eta)+\eta}(f) - \lceil \log k \rceil)$, *for* $0 < \varepsilon < 1$ *and* $0 < \eta < 1$. $\blacksquare$

(The proof is exactly the same as that for public coin protocols, excepting that at the beginning Alice randomly chooses the link i and sends the number i to Bob.) Apart from the loss in constant factors in case of ε close to $\frac{1}{2}$ respectively 1, we have a similar situation as in the nondeterministic case: there is no information about functions of small complexity on long arrays. Next, we derive lower bounds that do not involve the summand $\log k$.

The proof methods of Sections 3 and 4 seem not to be applicable here. This is due to the property of private coin protocols that inner processors are allowed to carry out their own random experiments and thus the proof of the bracket lemma is no longer valid. However, we obtain alternative bounds by invoking Newman's results on the relationship between public-coin complexity and private-coin complexity. The formulation is from [10].

Fact 6.4 [14] *If* $|X|, |Y| \leq 2^n$, *and* $f : X \times Y \to \{0,1\}$, *then*

(a) $R_0(f) = O\left(R_0^{\mathrm{pub}}(f)\right) + O(\log n)$;

(b) $R_{\varepsilon+\delta}(f) = R_\varepsilon^{\mathrm{pub}}(f) + O(\log n + \log \delta^{-1})$, *for* $0 < \varepsilon < \varepsilon + \delta < \frac{1}{2}$;

(c) $R^1_{\varepsilon+\delta}(f) = R_\varepsilon^{1,\mathrm{pub}}(f) + O(\log n + \log \delta^{-1})$, *for* $0 < \varepsilon < \varepsilon + \delta < 1$. $\blacksquare$

Theorem 6.5 *There are constants* $B, B_1, B_2 \geq 1$ *such that for any* $f : X \times Y \to \{0,1\}$ *with* $X, Y \subseteq \{0,1\}^n$ *we have the following:*

(a) $R_{0,k}(f) = \Omega(k \cdot (R_0(f) - B \log n))$;

(b) $R_{\varepsilon,k}(f) = \Omega(k \cdot (R_\varepsilon(f) - B_1 \log n - B_2 \log((\varepsilon(\frac{1}{2} - \varepsilon))^{-1})))$, *for* $0 < \varepsilon < \frac{1}{2}$;

(c) $R^1_{\varepsilon,k}(f) = \Omega(k \cdot (R_\varepsilon(f) - B_1 \log n - B_2 \log((\varepsilon(1-\varepsilon))^{-1}))$, *for* $0 < \varepsilon < 1$.

Proof. (a) This follows immediately by combining Observation 6.2(a) with Fact 6.4(a). (b) By combining Observation 6.2(b) with Fact 6.4(b) we obtain that for certain constants $C_1, C_2 \geq 1$ we have

$$R_{\varepsilon,k}(f) = \Omega(k \cdot R_\varepsilon^{\text{pub}}(f)) = \Omega(k \cdot (R_{\varepsilon+\delta}(f) - C_1 \log n - C_2 \log \delta^{-1})),$$

for arbitrary $0 < \varepsilon < \varepsilon + \delta < \frac{1}{2}$. For $\frac{1}{4} \leq \varepsilon < \frac{1}{4}$ we use $\delta = (\frac{1}{2} - \varepsilon)/2$ and the well-known fact $R_\varepsilon(f) = O(R_{\varepsilon/2+1/4}(f))$ to conclude that

$$R_{\varepsilon,k}(f) = \Omega(k \cdot (R_\varepsilon(f) - B_1 \log n - B_2 \log((\tfrac{1}{2} - \varepsilon)^{-1}))),$$

for suitable constants B_1, B_2. For $0 < \varepsilon < \frac{1}{2}$ we use $\delta = \varepsilon/2$. Clearly, we have $R_\varepsilon(f) = O(R_{3\varepsilon/2}(f))$ for these ε, and hence can conclude

$$R_{\varepsilon,k}(f) = \Omega(k \cdot (R_\varepsilon(f) - B_1 \log n - B_2 \log \varepsilon^{-1})).$$

(c) This is proved similarly as (b). ∎

7 One-tape Turing machines and communication complexity

It is a very natural idea that the time required to compute a function on a 1-tape Turing machine (without input or output tape) should have something to do with the communication complexity of that function. It is not a coincidence that for many of the functions for which time lower bounds on the Turing machine are known also communication complexity bounds are known.

We will find it convenient to consider an input format for the Turing machine in which the input is split into a left and a right part, which are separated by a "desert" or a "bridge" [17] of empty cells.

Notation. In the following, we only consider functions

$$f : \bigcup_{n \geq 1} (\{0,1\}^n \times \{0,1\}^n) \to \{0,1\}.$$

The restriction $f|\{0,1\}^n \times \{0,1\}^n$ is denoted by f_n. With $\hat{f}_{n,k}$ we denote the function from $\{0,1\}^n\{2\}^{k-1}\{0,1\}^n$ defined by $\hat{f}_{n,k}(x2^{k-1}y) = f_n(x,y) = f(xy)$.

7.1 Deterministic Turing machines

It is obvious (and has been observed many times before) that a deterministic Turing machine with an input format $x2^{k-1}y$ gives rise to a protocol for computing f_n on an array with k links, where the number of head movements determines the total length of the messages exchanged in the array. Thus, lower bounds for the array give rise to time lower bounds for the Turing machine.

The protocol for the array is constructed (informally) as follows. The tape is split into $k+1$ pieces, of which processor P_0 represents the part to the left of the bridge, $P_1, \ldots, P_{k-1}$ represent the single cells of the bridge, and P_k represents the cells to the right of the bridge. Communication rules for a processor are induced by the transition function of the Turing machine. We assume, without loss of generality, that at the end of the computation the Turing machine writes the result bit into all cells of the desert. (This guarantees condition (4) from Section 2.2.) It is easy to see that if the set of states Q of M is encoded in binary, then the array protocol can be made to use $\lceil \log_2 |Q| \rceil$ bits per Turing machine step. This implies the following bound.

Observation 7.1 *Assume a 1-tape Turing machine M with state set Q computes $\hat{f}_{n,k(n)}$, $n \geq 1$, for some sequence $k(n)$, $n \geq 1$, of natural numbers. If $time_M(x2^{k-1}y)$ denotes the running time of M on input $x2^{k-1}y$, then*

$$time_M(n) := \max\{time_M(x2^{k(n)-1}y) \mid x, y \in \{0,1\}^n\} \geq D_{k(n)}(f_n)/\lceil \log |Q| \rceil.$$

■

In combination with Tiwari's methods for proving bounds on $D_k(f)$ this observation yields lower time bounds for many important functions — those whose communication matrix has a large rank and those that admit large fooling sets. E.g., consider the distinctness problem: for a sequence x of numbers $x_1, \ldots, x_\nu$ and a sequence y of numbers $y_1, \ldots, y_\nu$ of μ bits each decide whether $x_i \neq y_i$ for all $1 \leq i \leq \nu$. Clearly, the assumption about the desert is inessential for this function. Using the well-known fact that the communication matrix of this function has rank $2^n = 2^{\nu\mu}$ (e. g., see [20]), we obtain that the complexity of this function on a 1-tape Turing machine is $\Omega(n^2) = \Omega((\nu\mu)^2)$. (For a related weaker lower bound, with a different proof, see [11].)

On the other hand, a longstanding open problem was to derive lower time bounds for functions in terms of $D(f_n)$; the obvious conjecture being $time_M(n) = \Omega(k(n) \cdot D(f_n))$. The best general lower bound was

$$time_M(n) = \Omega(k(n) \cdot (\sqrt{D(f_n)} - \log k(n))),$$

which is obtained through the detour via randomized zero-error protocols for the array, cf. Section 6. We show that the conjecture is true.

Theorem 7.2 *For arbitrary functions f and a sequence $k(n)$, $n \geq 1$, we have: if a one-tape Turing machine M computes $\hat{f}_{n,k(n)}$, for $n \geq 1$, then the worst case number of steps made by M on $x2^{k(n)-1}y$, $x, y \in \{0,1\}^n$, is $\Omega(k(n) \cdot D(f_n))$.*

Proof. The Turing machine computation induces a protocol for the array of length $k(n)$, in which the total number of bits sent is bounded by $\lceil \log |Q| \rceil$ times the number of head movements. This implies

$$\begin{aligned} D_{k(n)}(f_n) \quad \leq \quad & \lceil \log |Q| \rceil \cdot \text{maximal running time of } M \\ & \text{on an input from } \{0,1\}^n\{2\}^{k(n)-1}\{0,1\}^n. \end{aligned}$$

From Theorem 3.4 we know that $D_{k(n)}(f_n) \geq 0.146 \cdot k(n) \cdot D(f_n)$. ∎

Of course, the necessity of having a "desert" built into the input for the lower bounds to hold is annoying. For many functions, though, the communication complexity of $f_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ does not change by much if the $\frac{n}{2}$ last bits of the first part of the input and the first $\frac{n}{2}$ bits of the second part of the input are set constant, thus generating a situation with $\frac{n}{2}$ input bits on each side and a desert of size n. For the general case, we observe the following.

Theorem 7.3 *For arbitrary (computable) functions* $f : \bigcup_{n\geq 1}(\{0,1\}^n)^2 \rightarrow \{0,1\}$ *and all Turing machines* M *that compute* f *we have*

$$time_M(n) = \Omega(D(f_n)^2), \text{ for } n \geq 1,$$

where $time_M(n)$ *denotes the worst-case running time of* M *on an input of length* $2n$ *bits.*

Proof. Fix n, and let $k = k(n) = \lceil D(f_n)/4 \rceil$. For $\alpha, \beta \in \{0,1\}^k$ let $\hat{f}_{n,\alpha,\beta} : (\{0,1\}^{n-k})^2 \rightarrow \{0,1\}$ be the restriction of f obtained by fixing the rightmost k bits of x to the values given by α and the leftmost k bits of y to the values given by β. It is easily seen that there must exist choices for α, β so that $D(\hat{f}_{n,\alpha,\beta}) \geq \frac{1}{2}D(f_n)$. (Otherwise, we could obtain a cheap two-parity protocol for f_n as follows. On input (x, y), Alice sends the last k bits $\alpha(x)$ of x to Bob, who, in turn, sends the first k bits $\beta(y)$ of y to Alice. Then they run a protocol for $\hat{f}_{n,\alpha(x),\beta(y)}$ of complexity strictly smaller than $\frac{1}{2}D(f_n)$. This protocol has a complexity strictly smaller than $D(f_n)$, a contradiction.) We fix such α, β. Any Turing machine M that computes f computes $\hat{f}_{n,\alpha,\beta}$ with an input format that contains a desert of length $\frac{1}{2}D(f_n)$. As in the previous theorem we conclude that M makes $\Omega(D(f_n) \cdot D(\hat{f}_{n,\alpha,\beta})) = \Omega(D(f_n)^2)$ many steps on some input from $(\{0,1\}^n)^2$. ∎

7.2 Nondeterministic and randomized Turing machines

By the method mentioned in Observation 4.6 we immediately have that any nondeterministic one-tape Turing machine that recognizes $\hat{f}_{n,k(n)}$ for all $n \geq 1$ has time complexity $\Omega(k \cdot (N(f_n) - \log k))$. As in Section 7.1, we may show the lower bounds $\Omega(k(n) \cdot N(f_n))$ for computing $\hat{f}_{n,k(n)}$ and $\Omega(N(f_n)^2)$ for computing f. These bounds (with unspecified constants) were already obtained by Paturi and Simon [17], by totally different methods that can be applied only to Turing machines, not to processor arrays in general.

Turning to randomized Turing machines, we note that a very clever way of utilizing randomized communication complexity was used by Kalyanasundaram and Schnitger [7] to prove lower bounds for a special kind of problems (in particular, the element distinctness problem) on probabilistic one-tape Turing machines with an extra input tape. In [17], connections between randomized communication complexity and randomized Turing machines in the *unbounded error model* were established. (This is quite different from the bounded error or zero error model considered here.) Regarding the question whether in general lower bounds on communication complexity can be used to prove lower bounds on Turing machine complexity, the method from Proposition 6.2 could be used to obtain lower bounds like

$$R_0\text{-}time_1(\hat{f}_{n,k(n)}) = \Omega(R_0(f_n) - \log(k(n))),$$

or

$$R_\varepsilon\text{-}time_1(\hat{f}_{n,k(n)}) = \Omega(\delta \cdot R_{\varepsilon+\delta}(f_n) - \log(k(n))), \text{ for } 0 < \varepsilon < \varepsilon + \delta < \tfrac{1}{2},$$

where $R_0\text{-}time_1(\hat{f}_{n,k(n)})$ denotes the expected running time of an arbitrary probabilistic 1-tape Turing machine that recognizes $\hat{f}_{n,k(n)}$ with zero error, similarly for the bounded error cases.

These bounds are useless for large $k(n)$ and weak for ε that is very close to $\frac{1}{2}$. Our techniques, as described in section 6, make it possible to at least eliminate the dependence on ε. Moreover, the reader may recall that bounds like

$$\begin{aligned} R_0\text{-}time_1(\hat{f}_{n,k(n)}) &= \Omega(k(n) \cdot R_0^{pub}(f_n)); \\ R_\varepsilon\text{-}time_1(\hat{f}_{n,k(n)}) &= \Omega(k(n) \cdot R_\varepsilon^{pub}(f_n)); \\ R_\varepsilon^1\text{-}time_1(\hat{f}_{n,k(n)}) &= \Omega(k(n) \cdot R_\varepsilon^{1,pub}(f_n)) \end{aligned}$$

can always be used. ■

Acknowledgement. I would like to thank Pavol Ďuriš for helpful discussions during early stages of the work on the topic of this paper. Thanks are due to Juraj Hromkovič for motivating discussions during the long period this work lay half finished. A course taught by Ingo Wegener on the basis of the book manuscript by E. Kushilevitz and N. Nisan brought Fact 3.3, the "missing link" in the proof, to my attention.

References

[1] A. V. AHO, J. D. ULLMAN, and M. YANNAKAKIS, On notions of information transfer in VLSI circuits, in *Proc. of the 15th Annual ACM Symposium on Theory of Computing* (1983), pp. 133–139.

[2] B. CHOR and O. GOLDREICH, Unbiased bits from sources of weak randomness and probabilistic communication complexity, *SIAM J. Comput.* **17** (1988) 230–261.

[3] N. DERSHOWITZ and J.-P. JOUANNAUD, Rewrite systems, in J. VAN LEEUWEN, ed., *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, Elsevier, Amsterdam, 1990, pp. 243–320.

[4] B. HALSTENBERG and R. REISCHUK, Relations between communication complexity classes, *J. Comput. Syst. Sci.* **41** (1990) 402–429.

[5] J. HROMKOVIČ, *Communication Complexity and Parallel Computing*, EATCS Monographs on Computer Science, Springer-Verlag, Berlin, to appear.

[6] J. HROMKOVIČ and G. SCHNITGER, Nondeterministic communication with a limited number of advice bits, in *Proc. of the 28th Annual ACM Symposium on Theory of Computing* (1996), pp. 551–560.

[7] B. KALYANASUNDARAM and G. SCHNITGER, Communication complexity and lower bounds for sequential computation, in J. BUCHMANN ET AL., eds., *Informatik · Festschrift zum 60. Geburtstag von Günter Hotz*, B. G. Teubner, Stuttgart, 1992, pp. 253–268.

[8] M. KRAUSE and S. WAACK, Variation ranks of communication matrices and lower bounds for depth two circuits having symmetric gates with unbounded fan-in, in *Proc. of the 32nd IEEE Symposium on Foundations of Computer Science* (1991), pp. 777–782.

[9] E. KUSHILEVITZ, N. LINIAL, and R. OSTROVSKY, The linear-array conjecture in communication complexity is false, in *Proc. of the 28th Annual ACM Symposium on Theory of Computing* (1996), pp. 1–10.

[10] E. KUSHILEVITZ and N. NISAN, *Communication Complexity*, Cambridge University Press, Cambridge, to appear.

[11] A. López-Ortiz, New lower bounds for element distinctness on a one-tape Turing machine, *Inform. Proc. Lett.* **51** (1994) 311–314.

[12] K. Mehlhorn, *Data Structure and Algorithms I: Sorting and Searching*, EATCS Monographs on Computer Science, Springer-Verlag, Berlin, 1984.

[13] K. Mehlhorn and E. M. Schmidt, Las Vegas is better than determinism in VLSI and distributed computing, in *Proc. of the 14th Annual ACM Symposium on Theory of Computing* (1982), pp. 330–337.

[14] I. Newman, Private vs. common random bits in communication complexity, *Information Processing Letters*, **39** (1991) 67–71.

[15] M. H. A. Newman, On theories with a combinatorial definition of 'equivalence', *Ann. Math.* **43** (1942) 223–243.

[16] N. Nisan and A. Wigderson, On rank versus communication complexity, *Combinatorica* **15** (1995) 557–565.

[17] R. Paturi and J. Simon, Probabilistic communication complexity, *J. Comput. Syst. Sci.* **33** (1986) 106–123.

[18] A. A. Razborov, Applications of matrix methods to the theory of lower bounds in computational complexity, *Combinatorica* **10** (1990) 81–93.

[19] P. M. Spira, On time-hardware complexity tradeoffs for Boolean functions, in *Proc. of the 4th Hawaii Symposium on System Sciences*, 1971, pp. 525–527.

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

[21] A. C.-C. Yao, Some complexity questions related to distributed computing, in *Proc. of the 11th Annual ACM Symposium on Theory of Computing* (1979), pp. 209–213.

A Uniqueness of computations

This appendix gives the framework for a rigorous proof of the following fundamental lemma from Section 2. The proof will turn out to be relatively easy once the proper type of formalism is identified, which is related to the theory of term rewriting.

Lemma 2.1 *Let Φ be a deterministic protocol for inputs from $X \times Y$ on an array of length k. Then for each fixed input $(x, y) \in X \times Y$, any execution of Φ will generate exactly the same k message sequences $s_1, \ldots, s_k$ on links $1, \ldots, k$.*

Proof. The proof is based on properties (1), (2), and (3) from Section 2. We start by normalizing computations a little. We first eliminate link delays. By property (3) it cannot change the behaviour of processor P_i on link i, say, if more bits arrive on link $i+1$ or are sent across link $i+1$ by P_i. Thus, we may assume that all messages reach their destination instantaneously after being sent. This makes it possible to model intermediate states of the computation as sequences

$$(s_0, s_1, \ldots, s_k, s_{k+1}),$$

where $s_0 = x$, $s_{k+1} = y$, and for $1 \leq i \leq k$ the bit string s_i is a partial message sequence corresponding to a node of the protocol tree for link i. The computation proceeds in discrete time steps whose duration is of no importance. Before the first step, the state is $(s_0, \varepsilon, \ldots, \varepsilon, s_{k+1})$, and the computation ends in a state $(s_0, s_1, \ldots, s_k, s_{k+1})$ in which no more messages can be sent. One step consists in one or several messages being sent, which means that one or several of the s_i are extended by one bit, namely s_i may be extended by either $\Phi_i^+(s_{i-1}, s_i)$ or $\Phi_i^-(s_i, s_{i+1})$ according as P_i or P_{i+1} is allowed to speak next on link i. Next, we note that we can sequentialize steps in which several messages are sent in parallel, in an arbitrary order. Again, this is a consequence of rule (3): exchanging more information on one link will not change the bit sent next on the other link. Thus, we will assume that in one step exactly one of the s_i, $1 \leq i \leq k$, is extended by one bit, which has to be $\Phi_i^+(s_{i-1}, s_i)$ or $\Phi_i^-(s_i, s_{i+1})$. It remains to show that no matter in which order the possible moves allowed by the protocol trees and the $\Phi_i^\pm$-functions are executed, always the same final state is reached.

We first note that in a state $(s_0, s_1, \ldots, s_k, s_{k+1})$ reached at termination all s_i, $1 \le i \le k$, must correspond to leaves in their protocol tree. Indeed, suppose that is not the case. Form a directed graph with nodes $P_0, \ldots, P_k$ by drawing an arc from P_{i-1} to P_i if on the basis of s_i P_{i-1} is next to speak on link i, an arc from P_i to P_{i-1} if P_i is next. By assumption, this digraph has at least one arc, and obviously it has no cycles. Thus, there must be a source, i. e., a node only with outgoing arcs. This corresponds to a processor that either is next to speak on both its links or next to speak on one link, with the other link being finished. By rule (2) (no deliberate blocking) it must be possible for P_i to send at least one message, thus the computation cannot stop in state $(s_0, s_1, \ldots, s_k, s_{k+1})$.

We will now employ an argument that is basic in the context of term rewriting and string rewriting for proving "confluence" of a set of rewrite rules (see [3]). Namely, we observe that our system of states and rules is finite (since all protocol trees are finite), acyclic (since a step increases the total length of the message sequences $s_0, s_1, \ldots, s_k, s_{k+1}$) and "locally confluent", meaning the following:

- if $s' = (s_0, s'_1, \ldots, s'_k, s_{k+1})$ and $s'' = (s_0, s''_1, \ldots, s''_k, s_{k+1})$ can be reached from $s = (s_0, s_1, \ldots, s_k, s_{k+1})$ by performing one step, then there is a state $\hat{s} = (s_0, \hat{s_1}, \ldots, \hat{s_k}, s_{k+1})$ that can be reached from both s' and s'' by performing an appropriate sequence of steps.

Indeed, assume that s' is obtained from s by extending s_i by a bit $b' = \Phi_i^+(s_{i-1}, s_i)$, say, and s'' is obtained from s by extending s_j by a bit $b'' = \Phi_j^+(s_{j-1}, s_j)$ or $b'' = \Phi_j^-(s_j, s_{j+1})$. Clearly, if $i = j$, then b' and b'' must be the same, since it must be P_{j-1} that speaks. If $j = i - 1$, then after extending s_{i-1} to $s''_{i-1} = s_{i-1}b''$ we may still send b' across link i, since $\Phi_i^+(s_{i-1}, s_i) = \Phi_i^+(s_{i-1}b'', s_i)$ by rule (3). Similarly, b' may be sent across link i in state s''. Thus, we can reach the state $\hat{s} = (s_0, s_1, \ldots, s_{i-1}b'', s_ib', \ldots, s_k, s_{k+1})$ from both s' and s''. If $j = i + 1$ and $b'' = \Phi_j^-(s_j, s_{j+1})$ then processor P_i sends bits b' (across link i) and b'' (across link $i+1$). Again, by rule (3), this can be done in either order. Finally, if $j \notin \{i-1, i, i+1\}$, then there is no conflict at all between the processors involved in sending b' and b'', which means that the order does not matter. Applying a principle known as Noetherian induction ([3, p. 267], [15]) we may conclude from local confluence that our system is "globally confluent", which means the following:

- if s' and s'' can both be reached from s by performing a sequence of steps, then there is a state $\hat{s}$ that can be reached from both s' and s'' by applying a sequence of steps.

This property implies that no matter how our computation proceeds, starting from $(s_0, \varepsilon, \ldots, \varepsilon, s_{k+1})$, the terminal state that is finally reached must be the same. (If two different terminal states could be reached, which cannot be extended, we would have an immediate contradiction to the confluence property.) This, of course, means that the sequence of bits exchanged on each link does not depend on the order in which the computational steps are executed, which is the assertion of the lemma. ∎

B Depth versus number of message sequences

This appendix provides the proof of a folklore fact that relates the logarithm of the minimal number of leaves in a protocol tree for f to the deterministic communication complexity $D(f)$. The proof is almost the same as in [10], and is provided here only for the convenience of the reader.

Fact 3.3 *If f has a two-party protocol with at most $c \geq 1$ different message sequences (i.e., the protocol tree has c leaves), then f has a two-party protocol of complexity at most $2\log_{3/2} c$.*

Proof. For $c \geq 1$, let

$$d(c) = \max\{D(f) \mid f \text{ admits a protocol tree with } c \text{ or fewer leaves}\}.$$

We show, by induction, that $d(c) \leq 2\log_{3/2} c$. — Clearly, $d(1) = 0 = \log_{3/2} 1$. Now let $c \geq 2$, and consider any function f that admits a protocol tree T with c or fewer leaves. It is easy to see that this tree has a node v such that the subtree T_v rooted at v has at least $c/3$ leaves and at most $2c/3$ leaves. If v happens to be the root, then $T = T_v$ has no more than $\lfloor 2c/3 \rfloor$ leaves, hence, by induction, $D(f) \leq 2\log_{3/2}\lfloor 2c/3 \rfloor \leq 2\log_{3/2} c$. Thus we may assume that v is not the root. Recall that the set of inputs that reach v when the protocol is performed is a rectangle $R_v = X_v \times Y_v$ with $X_v \subseteq X$ and $Y_v \subseteq Y$. We describe a new protocol. By exchanging two bits, Alice and Bob decide whether (x, y) is in R_v or not. If so, they run a protocol of optimal depth at most $d(\lfloor 2c/3 \rfloor)$ for the function computed by the protocol described by T_v. (This gives correct results on inputs from R_v.) If $(x, y) \notin R_v$, they run a protocol of optimal depth at most $d(\lfloor 2c/3 \rfloor)$ for the function computed by the tree resulting from T by discarding T_v and placing the sibling of v at the position of the parent of v. (It is easily seen that this gives correct results on inputs from $X \times Y - R_v$.) The complexity of the whole protocol is bounded by $2 + d(\lfloor 2c/3 \rfloor)$, which, by induction, is at most $2 + 2 \cdot \log_{3/2}(2c/3) = 2\log_{3/2} c$.

■