INFORMATION AND COMPUTATION 114, 214–236 (1994)

Computing Boolean Functions on Anonymous Networks*

EVANGELOS KRANAKIS AND DANNY KRIZANC

School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada, and Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

AND

JACOB VAN DEN BERG

Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

We study the bit-complexity of computing Boolean functions on anonymous networks. Let N be the number of nodes, δ the diameter, and d the maximal node degree of the network. For arbitrary, anonymous networks we give a general algorithm of polynomial bit complexity $O(N^3 \cdot \delta \cdot d^2 \cdot \log N)$ for computing any Boolean function which is computable on the network. This improves upon the previous best known algorithm, which was of exponential bit complexity $O(d^{N^2})$. For symmetric functions on arbitrary networks we give an algorithm with bit complexity $O(N^2 \cdot \delta \cdot d^2 \cdot \log^2 N)$. This same algorithm is shown to have even lower bit complexity for a number of specific networks, for example tori, hypercubes, and random regular graphs. We also consider the class of distance regular unlabeled networks and show that on such networks symmetric functions can be computed efficiently in $O(N \cdot \delta \cdot d \cdot \log N)$ bits.

---

* A preliminary version of this article appeared as an extended abstract in "Proceedings of the 17th International Colloquium on Automata, Languages and Programming" (M. Paterson, Ed.), Lecture Notes in Computer Science, Vol. 443, Springer-Verlag, Berlin/New York, 1990.

0890-5401/94 $6.00

1. INTRODUCTION

A main goal of distributed computing is the design and analysis of efficient algorithms for computing interesting functions that run on distributed networks of processors. A basic assumption in this analysis is that the efficiency of such algorithms is limited by the amount of interprocessor communication required by the algorithm. In this paper we attempt to quantify how much communication effort is required to compute a simple class of functions on a distributed network of processors. The model of a distributed network we consider is that of an *anonymous network*. (The model will be formally defined in Section 2.) This model has been well-studied in the literature [4, 5, 14, 17, 24, etc.], since it embodies a small number of assumptions while still admitting nontrivial solutions to the questions raised. It allows us to study the effect of symmetry (of the network and of the functions to be computed) on distributed computations.

Informally, by an anonymous network we understand a connected graph. The nodes of the graph represent the processors of the network and the edges represent the available communication links between processors. The processors are entirely homogenous, having no unique identites (hence the name anonymous). In a computation, each processor starts in some initial state with some input, proceeds by sending and receiving messages through the network links according to some deterministic algorithm, and ends its computation after a finite number of steps by entering into a final state. An action of a processor depends on its current state and the input data to its algorithm. Moreover, the processors execute the same action given the same state and input data. At the end of the computation each processor outputs the (same) desired function of the inputs.

For simplicity, we will consider the problem of computing the class of Boolean functions, i.e., functions with $\{0, 1\}$-valued inputs and outputs. This class contains a number of interesting and important examples (OR, AND, XOR, etc.) and is rich enough to capture the notions studied here. It is relatively straightforward to extend the algorithms described below to more general functions (at a corresponding cost in complexity). The communication cost incurred during a computation is measured in terms of the number of bits exchanged by the processors, termed the *bit complexity* of the computation. We prefer to study the bit complexity rather than the message complexity (the number of messages exchanged by processors) so as not to hide the costs associated with sending large messages. In some sense, the message complexity measures the number of communication rounds required to compute a function, whereas the bit complexity measures the total amount of information that must be exchanged to compute a function. In the algorithms below the sizes of the messages sent is

clear from the context, and thus it is easy to calculate the message complexity of the algorithm.

1.1. *Previous Work*

In the sequel we assume that N is the number of processors of a given anonymous network.

The simplest topology considered in the study of the bit complexity of computing Boolean functions is the ring, e.g., [1, 3, 4, 17, 19]. It has been shown by Attiya *et al.* [4] that there is an algorithm for computing all Boolean functions which are computable on the ring, with bit complexity $O(N^2)$. In addition, Moran and Warmuth [17] show that any nonconstant function has bit complexity $\Omega(N \cdot \log N)$ on the ring, and also construct Boolean functions with bit complexity $\Theta(N \cdot \log N)$ on the ring. For the anonymous torus network Beame and Bodleander [5] give an algorithm with bit complexity $O(N^{1.5})$, and construct nonconstant functions with bit complexity $\Theta(N)$. In the case of the anonymous hypercube network Kranakis and Krizanc [14] give an algorithm with bit complexity $O(N \cdot \log^4 N)$. (In [5, 14] the networks are assumed to have a globally consistent labelling on the edges.)

For general networks Yamashita and Kameda [23, 24] show that the message complexity of computing a Boolean function on an arbitrary anonymous network is $O(N^2 \cdot m)$, where m is the number of links of the network. However, these messages consist of trees of depth N^2 with fanout corresponding to the degrees of the nodes of the network. For regular graphs of degree d this translates into an exponential $O(d^{N^2})$ bit complexity ($d = 4$ for the torus, and $d = \log N$ for the hypercube).

1.2. *Results of the Paper*

In the present paper we study the bit complexity for Boolean functions on arbitrary anonymous networks and on distance regular anonymous networks (see Section 5 for the definition of distance regular). We show in Section 3 that for any anonymous N-node network of maximal node valency d and diameter δ, every Boolean function which is computable on the network can be computed in $O(N^3 \cdot \delta \cdot d^2 \cdot \log N)$ bits. This significantly improves the previous $O(d^{N^2})$ upper bound of [23]. In Section 4 we give an algorithm for computing symmetric functions with bit complexity $O(N^2 \cdot \delta \cdot d^2 \cdot \log^2 N)$. This same algorithm provides even more efficient algorithms when applied to specific networks, like tori, hypercubes, and random regular graphs. For the case of distance regular networks we show in Section 5 how to compute any symmetric function in $O(N \cdot \delta \cdot d \cdot \log N)$ bits. We conclude in Section 6 with some discussion and open problems.

2. Model of Computation

A distributed network is a simple, connected graph consisting of nodes (vertices) on which the processors are located, and links (edges) along which the interprocess communication takes place. The processors are assumed to have unlimited computational power but may exchange messages only with their neighbors in the network. Initially, each processor is given an input bit, 0 or 1.

The processors follow a deterministic protocol (or algorithm). During each step of the protocol they perform local computations depending on their input value, their previous history, and the messages they receive from their neighbors. Then they transmit the result of this computation to some or all of their neighbors. After a finite number of steps, predetermined by the initial conditions and the protocol, the processors terminate their computation and output a single bit.

Let B_N be the set of Boolean functions on N (Boolean) variables. Let $\mathcal{N} = (V, E)$ be a network of size N, with node set $V = \{0, 1, ..., N-1\}$ and edge set $E \subseteq V \times V$. An input to $\mathcal{N}$ is an N-tuple $I = \langle b_v : v \in V \rangle$ of bits $b_v \in \{0, 1\}$, where processor v receives as input value the bit b_v. Given a function $f \in B_N$ known to all the processors in the network we are interested in computing the value $f(I)$ on all inputs I. To compute f on input $I = \langle b_v : v \in V \rangle$, each processor $v \in V$ starting with the input bit b_v should terminate its computation according to the given protocol. A network computes the function f if for each input I, at the end of the computation each processor computes the value $f(I)$.

The bit complexity for computing f is the total number of bits exchanged during the computation of f. We are interested in providing algorithms that minimize the bit complexity of computing Boolean functions.

We assume that the processors transmit messages through the links in finite time and are error-free. In addition, we make the following assumptions regarding the networks and their processors:

1. The processors know the network topology and the size of the network (i.e., total number of processors).

2. The processors execute identical algorithms. In particular, this implies that the processors are anonymous (i.e., they do not know the identities of themselves or of the other processors).

3. The processors are deterministic.

4. The network is asynchronous.

5. The processors can distinguish their communication links; i.e., the links are given labels which are known to the processors locally.

However, there is no global, consistent labeling (or orientation) of the network links.

6. The network links are FIFO.

The class of computable functions under the above model depends on the given network. Although it will turn out that all symmetric boolean functions are computable on any network, there may exist functions which are not computable on specific networks. For example, take N processors arranged on an oriented ring. A Boolean function which changes value when we rotate the inputs cyclically is not computable (e.g., $f(x_1, x_2, ..., x_N) = x_1$ is such a function). Thus in the case of oriented rings the processors must be given a boolean function which satisfies the invariance condition $f(x_1, x_2, ..., x_N) = f(x_N, x_1, x_2, ..., x_{N-1})$ on all inputs.

Intuitively speaking, the above assumptions will imply that a function is computable in a network if and only if its output depends not on the inputs themselves but rather on the similarity class of the inputs (a precise formulation of this statement will be given later in Theorem 5). For particular labeled networks (rings, hypercubes, Cayley networks, etc.) simpler characterizations are known which depend only on the automorphism group of the network. For more on these topics consult [13, 14].

Observe that our algorithms are valid on any asynchronous network provided that the links are FIFO. It makes no difference which processor "starts" as long as they complete the "send–receive" cycle in the order specified by the protocol. Moreover, a flood of wake-up messages would cost only $O(|E|)$ bits, where E is the number of links of the network.

Note that changing any of the above assumptions changes the computational capabilities and limitations of the model. If the size of the network is not known to the processors then it may not even be possible to compute any nonconstant function, e.g., in the ring [4]. Angluin [2] has shown that if the processors are anonymous and identical there is no algorithm for electing a leader. Yamashita and Kameda [24] have studied the effect of not knowing the topology of the network. If we add randomization to the model it becomes possible to improve greatly the average and worst case bit complexity. In synchronous networks information can be gathered not only through message passing but also through the absence of communication during a particular time interval. Labeling the edges of a network can be shown to change the set of functions computable on the network (see [14]).

In general, there are a number of models that could seemingly improve our results. Nevertheless, this would have to be at the cost of relaxing some of our rules regarding the deterministic execution of the protocols,

distributivity of computation, and asynchronicity of communication. This also seems to indicate that more efficient algorithms would presumably need additional global information.

3. Arbitrary Boolean Functions

In this section we give a general algorithm which computes any Boolean function computable on a given network using polynomial bit complexity. Before presenting the main algorithm we present some introductory results on idempotent operations.

3.1. *Algorithm for Idempotent Operations*

One of the results that will be used very frequently in the sequel concerns the computation of certain simple operations, such as maximum and set-union on general anonymous networks. To facilitate and simplify our discussion and avoid unnecessary repetition we state our main algorithm for computing such functions as a separate theorem. First we need a few definitions.

Let $\diamond$ be a commutative, associative, and idempotent binary operation on a set A; i.e., $\diamond: A \times A \to A$ satisfies the following axioms for all a, b, $c \in A$:

- $\diamond(a, b) = \diamond(b, a)$ (commutativity),
- $\diamond(a, \diamond(b, c)) = \diamond(\diamond(a, b), c)$ (associativity),
- $\diamond(a, a) = a$ (idempotency).

Such operations include maximum, minimum, set-union, and set-intersection. For simplicity, from now on we will abbreviate $\diamond(a, b)$ by $a \diamond b$.

Let $\mathcal{N} = (V, E)$ be an anonymous network and let $\diamond$ be an operation satisfying the previous three conditions. Let A^N be the set of all N-tuples from elements of A. For any input $I = \langle i_p : p \in V \rangle \in A^N$ to the network we can define a function $\diamond: A^N \to A$ by the following equation:

$$\diamond(I) = i_0 \diamond i_1 \diamond \cdots \diamond i_{N-1}.$$

(By an abuse of notation we use the same symbol for the binary operation $\diamond: A \times A \to A$ and the function $\diamond: A^N \to A$.) In view of the associativity of $\diamond$ this function is well defined. As a first step in our goal of providing an algorithm for computing all (computable) Boolean functions we will show that functions, like $\diamond$, which arise from such binary operations give rise to computable functions. The algorithm presented below was independently discovered by Tel [22] in the context of infimum computations in directed networks.

THEOREM 1. *Let $\mathcal{N}$ be an anonymous network with maximal node valency d and diameter δ and let $\diamond$ be a commutative, associative and idempotent binary operation. There is an algorithm for computing $\diamond(I)$ for any input $I = \langle i_p : p \in V \rangle \in A^N$ with bit complexity $O(N \cdot \alpha \cdot \delta \cdot d)$, where α denotes the number of bits necessary to represent an element of A.*

Proof. The idea of the algorithm is rather simple. Each processor sends its initial input value to all its neighbors. After receiving a value from its neighbors it applies the operation $\diamond$ to the value it already has and the values it receives. Every processor executes these steps δ many times. Eventually every input value to a node of the network will be distributed and accounted for by every other processor.

More formally the algorithm is as follows. Let $I = \langle i_p : p \in V \rangle$ be the input to the network.

Algorithm for processor p:
Initialize: $value_p[0] := i_p$;
for $i := 0, 1, ..., \delta - 1$ **do**
  send $value_p[i]$ to all neighbors of p;
  receive $value_q[i]$ from all neighbors q of p;
  compute $value_p[i+1] :=$
  $\diamond(\{value_p[i]\} \cup \{value_q[i] : q \text{ is a neighbor of } p\})$;
od;
output $value_p := value_p[\delta]$.

The proof of correctness of the algorithm is not difficult. Since the "send" procedure of the algorithm is iterated δ times (where δ is the diameter of the network) it is clear that the algorithm guarantees that all input values are taken into account when calculating the value of the function. By commutativity and associativity it is immaterial in what order the opertion $\diamond$ is applied to the given values. It can happen that in the course of the execution of the above algorithm by processor p the operation $\diamond$ is applied more than once to some element a, which is the initial input value to a certain processor q. The number of times $\diamond$ is applied depends on the number of walks of length less than δ from p to q through the network. However, because of the idempotency of the operation $\diamond$, we have that $a \diamond a \diamond \cdots \diamond a = a$. It follows that all processors will compute exactly the same value $\diamond(I)$, namely $value_p = \diamond(I)$, for all p.

It remains to determine the bit complexity of the algorithm. The processors receive through their neighbors elements of A, apply the operation $\diamond$, create new elements of A, and transmit them to their neighbors. The cost of transmitting each of these elements is α, the number of bits necessary to represent an element of A. Each of the N processors transmits a value to its d neighbors once in each of the δ phases of the above algorithm. This gives the desired bit complexity. ∎

An obvious corollary of the theorem concerns the bit complexity of the OR_N function. Since binary OR_N is a commutative, associative, and idempotent operation, Theorem 1 implies the following result, which will be useful later.

COROLLARY 2. *On an anonymous N-node network with maximal node valency d and diameter δ the OR_N function can be computed with bit complexity $O(N \cdot \delta \cdot d)$.* ∎

A simple extension of the lower bound for OR_N on the ring in [4]shows that if the network is regular then OR_N requires $\Omega(N \cdot \delta)$ bits to compute. Thus for this case the above algorithm is optimal to within a factor of d.

For any input $I = \langle i_p : p \in V \rangle \in A^N$ to the network let $\{i_p : p \in V\}$ denote the input-set corresponding to the input I. The next corollary concerns the bit complexity of computing the input-set for a given input and will be useful in the proof of our general Theorem 6 about the bit complexity of computable Boolean functions on general networks.

COROLLARY 3. *Let $\mathcal{N}$ be an anonymous N-node network with maximal node valency d and diameter δ. There is an algorithm for computing the input-set $\{i_p : p \in V\}$ of any input $I = \langle i_p : p \in V \rangle \in A^N$ with bit complexity $O(N^2 \cdot \alpha \cdot \delta \cdot d)$, where α denotes the number of bits necessary to represent an element of A.*

Proof. Here we apply the main Theorem 1 to the binary operation union, $\diamond(a, b) = a \cup b$, where the input to node p is the singleton set $\{i_p\}$. The elements transmitted in the course of the algorithm are subsets of the set $\{i_p : p \in V\}$. Each element can be coded with α bits, and therefore such sets can be coded with $N \cdot \alpha$ bits. ∎

3.2. *A Digression on Views and Computability*

Before proceeding with a discussion of our main algorithm it will be necessary to digress for a moment in order to present some prerequisites from [23] concerning the construction of "views" by processors and the computability of Boolean functions.

As in Angluin [2] and Yamashita and Kameda [23], we assume that the processors are given labels for their corresponding links. By this we mean an a priori numbering on the links of each processor. The numbering of each node is done locally and independently of the others. If necessary, each processor p renames all its incident links with the numbers $1, 2, \ldots, \deg(p)$, where $\deg(p)$ is the degree of p. Let us denote this labeling $\mathcal{L}$. Further assume that the input configuration $I = \langle b_p : p \in \mathcal{N} \rangle$ is assigned to the processors of the network, with bit b_p assigned to processor p.

The view of processor p with respect to the input configuration I and labeling $\mathscr{L}$, denoted by $T^{\infty}_{\mathscr{L},I}(p)$, is an infinite, labeled rooted tree defined recursively in the following way. The label of the root of $T^{\infty}_{\mathscr{L},I}(p)$ is the input bit b_p; for each processor p_i, $i=1, ..., \deg(p)$, adjacent to p the tree $T^{\infty}_{\mathscr{L},I}(p)$ has a vertex p_i and an edge from the root to p_i labeled $l(p, p_i) := \{\mathscr{L}(p, p_i), \mathscr{L}(p_i, p)\}$. Moreover, at the vertex p_i of the tree $T^{\infty}_{\mathscr{L},I}(p)$ is rooted the tree $T^{\infty}_{\mathscr{L},I}(p_i)$.

Two views are called similar if they are isomorphic, including edge and vertex labels. It is clear that this similarity is an equivalence relation all of whose equivalence classes have the same size (see [23]). We also denote by $T^{h}_{\mathscr{L},I}(p)$ the tree obtained from $T^{\infty}_{\mathscr{L},I}(p)$ by removing all levels of height $>h$.

An important consideration in the study of the complexity of our algorithms concerns the height of the tree which is sufficient in order that each processor can retrieve the view of all the processors in the network from its own view. This can be resolved by the fact that "isomorphism up to depth N^2 implies isomorphism up to all depths." For a proof the reader should consult [23].

More recent investigations by Norris allow an even stronger version of this last result, which in turn implies a factor N improvement in the complexity of our main algorithm. In order to obtain the optimal possible result we state (without proof) the main result of Norris [18], that "isomorphism up to depth $N-1$ implies isomorphism up to all depths."

LEMMA 4. *If* $T^{N-1}_{\mathscr{L},I}(p)$, $T^{N-1}_{\mathscr{L},I}(p')$ *are similar, so are* $T^{\infty}_{\mathscr{L},I}(p)$, $T^{\infty}_{\mathscr{L},I}(p')$. ∎

The second issue to concern us in this section is the class of Boolean functions which are computable in the network. The notion of computability on anonymous networks, which was defined in Section 2, involved precise rules on the behavior of the processors and the required interprocess communication. Theorem 4.1 of [23] connects our notion of computability of a Boolean function with its invariance under a suitable class of automorphisms of the network. The exact result is the following.

THEOREM 5. *A Boolean function f is computable in a network $\mathscr{N}$ if and only if, for any inputs I, I' such that there exist labelings $\mathscr{L}$, $\mathscr{L}'$ for which the trees $T^{\infty}_{\mathscr{L},I}$, $T^{\infty}_{\mathscr{L}',I'}$ are similar, we must have that $f(I)=f(I')$.* ∎

The algorithms described below are for Boolean functions which are computable on the given anonymous network.

3.3. *General Algorithm*

We are now ready to give our algorithm for computing arbitrary Boolean functions on a given anonymous network. We will prove the following theorem.

THEOREM 6. *Let $\mathcal{N}$ be an anonymous N-node network with maximal node valency d and diameter δ. There is an algorithm that computes any Boolean function which is computable on the network with bit complexity $O(N^3 \cdot \delta \cdot d^2 \cdot \log N)$.*

Proof. Let $f \in B_N$ be any Boolean function, known to all processors, which is computable on the anonymous network $\mathcal{N}$. Let $I = \langle b_p : p \in V \rangle$ be the input to the network, where b_p is the input to node p. Before proceeding with the formal description we outline the intuition behind the algorithm.

Overview of the Algorithm. The algorithm has two phases. In the first phase each node "unwraps" the network and forms a view (a labeled tree with itself as root). During this unwrapping procedure the processors exchange messages by executing an algorithm that enables them to "form" a view of what the input to the network should be. This is the most expensive part of the algorithm and two important complexity issues need to be taken into account: the first concerns the number of iterations of the unwrapping procedure required for the processors to receive sufficient information, and the second the number of bits required to code the information transmitted throughout the computation in order to minimize the number of bits exchanged. We take care of the first requirement by applying Lemma 4, and the second by developing a simple coding algorithm that economizes on the number of transmitted bits by encoding prior to transmission. Naturally, the coding and decoding algorithms require computation, but this is local to the processors and does not contribute to the overall bit complexity.

After the end of phase one, each processor will have access to a full view of the network. In phase two each processor will be able to compute a configuration I' (which is not necessarily identical to the input configuration I) but which corresponds to a view of the network "similar" to the one provided by the view I. Different processors may compute different views but they can all agree on a common configuration, say I', by exchanging messages and using Theorem 1. Since the function f is computable its output depends not on the input but rather on the similarity class of the views. It follows that $f(I') = f(I)$, and all processors output the value $f(I')$ which is also the correct value.

Main Algorithm. Our algorithm relies on several cost efficient adjustments and improvements of the algorithm of [24], using Theorem 1. Here is a formal presentation of the two phases.

As indicated in Section 3.2, we assume that the links associated with processor p are labeled with the numbers 1, 2, ..., $\deg(p)$. Now each processor transmits to each neighbor the label it has chosen for the link connecting them. Let $\mathscr{L}$ be the resulting labeling of the network $\mathscr{N}$. Next, each pair (p, q) of processors label their corresponding link with the multiset

$$l(p, q) = \{\mathscr{L}(p, q), \mathscr{L}(q, p)\}.$$

The processors keep this labeling fixed throughout the algorithm. It should be pointed out that this is only a local labeling and not a global orientation of the network; the processors know only the labeling of their corresponding links, and are completely unaware of the choice of labeling by the other processors in the network.

Phase 1. In this phase each processor gathers information from the rest of the processors about the input to the network in order to be able to compute correctly the value $f(I)$. Each processor p computes its view, $T_{\mathscr{L},I}(p)$. Since $\mathscr{L}$ and I are fixed below we will denote the view of p by T_p. This is a vertex and edge labeled tree of depth N. In a sense, each node p "unwraps" the network and forms a tree with itself as root. Since the network is anonymous it cannot use names for the processors; instead, it can only label the vertices of the tree with the input bits it receives in the course of the interprocess communication. Thus, the root of T_p is labeled with the input bit b_p and the node corresponding to q is labeled with the bit b_q. However, it needs to be stressed here that when the processors label a node with the bit b_q they do not necessarily know that the name of the processor they are labeling is q. (See Subsection 3.2 for details.)

The processors need to exchange enough information in order to compute correctly each T_p. They do this by exchanging the views they have constructed. However, trees of depth i and branching d have exponential bit complexity $\Omega(d^i)$ and transmitting them is expensive. Therefore we must be careful if we want to achieve an algorithm with polynomial bit complexity.

The novel idea here is that the processors do not need to transmit the whole trees. Instead they proceed in a level-by-level computation of the views. Assuming that at level i the processors have computed their trees up to depth i, they can use the algorithm given in Corollary 3 to compute the set of "encoded" trees. (Note that this set will be the same for all processors.) Since at each level there are no more trees than there are processors in the network, the codes can be taken to be integers $\leqslant N$. Now the processors proceed to the next level $i+1$ transmitting trees of height 1.

They use these encoded trees as leaves and leave it to the recipient processors to decode the information (which they can do since they have all computed the previous set of trees).

In the sequel we concentrate on the issue of coding and transmission of the trees concerned. Processor p computes a sequence of trees T_p^i of depth i, $i = 0, 1, ..., N-1$, by executing the following algorithm.

Algorithm for processor p:
Initialize: $T_p^0 := b_p$ **and** $set_p^0 := \{T_p^0\}$;
for $i := 0, ..., N-1$ **do**
  compute the set $set_p^i := \{T_q^i : q \in V\}$;
  code the elements of the set set_p^i with integers $1, ..., k$,
  where $k \leqslant N$ is the number of elements set_p^i,
  by ordering the set set_p^i lexicographically and letting
  $code(T_q^i) = j$, if T_q^i is the jth tree in this ordering;
  form the tree T_p^{i+1}; it is a tree of depth 1 with root labeled b_p;
  for each neighbor q of p there is an edge labeled $l(p, q)$;
  its leaves are labeled $cod(T_q^i)$, where q is a neighbor of p;
  send the tree T_p^{i+1} to all the neighbors of p;
od;
output set_p^{N-1}.

After the trees of level i have been constructed the processors use the set algorithm given in Corollary 3 to compute the set $\{T_p^i : p \in V\}$. Once all processors know all the trees of depth i there is no need to transmit to each other the decoded full trees themselves. It is sufficient to transmit the codes of the trees, and these can be just integers from 1 up to N, provided that they all perform "identical" algorithms that will enable them to decode the trees and subsequently generate the views.

To code the trees the processors order them lexicographically and let the code of the tree T be j, if T is the jth tree in this ordering. The processors then form new trees of depth $i+1$, namely T_p^{i+1}, by using these codes as leaves. Namely, the tree has a root which is labeled with p's input bit. The leaves of the tree consist of the above codes of the corresponding trees of depth i and the edges have the corresponding labeling. Now the processors transmit these new trees to all their neighbors, etc. To decode the trees the recipient processors need only look at their leaves in order to derive the whole view up to the current level. As indicated above we iterate this algorithm N times.

Phase 2. At this point all processors have computed the set of all views of depth $N-1$, namely the set $\{T_p^{N-1} : p \in V\}$. As explained in Subsection 3.2 we define an equivalence relation among trees. Two trees T and T' are equivalent if they are isomorphic including vertex and edge labels, but ignoring names of the vertices. By Lemma 4 for any two trees if their

restrictions to depth $N-1$ are isomorphic then the full trees themselves must also be isomorphic. Let $[T]_{I,\mathscr{L}}$ denote the equivalence class of T, where the subscript is to stress the dependence of the equivalence class on the input and the chosen labeling. It follows from the above discussion that each processor will be able to find representatives of all the equivalence classes of the full trees. The conditions imposed on the processors, and stated in the Introduction, regarding anonymity and the fact that all processors execute the same algorithm given the same state and input data guarantees that the Boolean function f is not sensitive to automorphisms applied on the input configuration. It follows that since f is computable on the network its value depends only on the equivalence classes of the trees above; i.e., for any inputs I, I' and any labelings $\mathscr{L}$, $\mathscr{L}'$, if $[T]_{I,\mathscr{L}}=[T']_{I',\mathscr{L}'}$, for any trees T, T', then $f(I)=f(I')$ (see also [23, Theorem 4.1]).

Now the processors want to compute $f(I)$, but they do not know the input I. To resolve this problem each processor

1. retrieves all the equivalences classes of the views in the network from its view (this can be done in view of Lemma 4), and

2. uses its knowledge of the network topology to construct a labeling $\mathscr{L}'$ and an input I' such that $[T]_{I,\mathscr{L}}=[T]_{I',\mathscr{L}'}$, for all trees T.

Certainly, each processor may choose a different input I' and labeling $\mathscr{L}'$. However, by exchanging information using Corollary 3, the processors can agree on a unique input I' and labeling $\mathscr{L}'$ based on some predetermined order. Since f is computable in the network its value depends only on the equivalence classes of the trees. We conclude that $f(I)=f(I')$. Thus it is sufficient to output $f(I')$ and this will be the desired, correct value assumed by f on input I.

This concludes the description of the algorithm. It remains to determine its bit complexity.

Complexity. Both phases involve local computations which do not require any bit exchanges. The application of Corollary 3 in phase 2 is easily estimated to be $O(N^3\cdot\delta\cdot d)$, which is well within the permissible bound of our theorem. The main bit exchanges take place in phase 1. There we have N iterations of the algorithm in Corollary 3. We need $d\cdot\log N$ bits to represent each of the corresponding trees. (The maximal degree of a network node is d and this same d is the maximal branching of the trees. The encoded trees are of depth 1 and their leaves are the encoded "subtrees" which are integers $1,\ldots,k$, for some $k\leqslant N$.) This means that the bit complexity of phase 1, and hence also of the algorithm as a whole, is $O(N^3\cdot\delta\cdot d^2\cdot\log N)$. ∎

4. Symmetric Functions

Symmetric functions are easier to compute because their output depends only on the number of 1's present in the input. As expected there exists a much more efficient algorithm for this class of functions. In this section we give an algorithm which computes any symmetric function on an anonymous network, improving upon the algorithm given above in this case. Our algorithm is deterministic but we use techniques from Markov chains in order to study the termination characteristics of our protocol and deduce its complexity.

4.1. *Algorithm for Symmetric Functions*

By the weight of a sequence $I=(b_1, ..., b_N)$ of bits we understand the number of b_i's which are equal to 1. Let $\mathcal{N}=(V, E)$ be an anonymous network of size N, with node set $V=\{0, 1, ..., N-1\}$ and edge set $E \subseteq V \times V$. To simplify the analysis below we will consider the network $\mathcal{N}'=(V, E \cup \{(i, i) \mid i=0, ..., N-1\})$ (i.e., $\mathcal{N}$ with self loops added to each vertex). Let $\deg(i)$ and $\deg'(i)=\deg(i)+1$ be the valency of node i in $\mathcal{N}$ and $\mathcal{N}'$, respectively. Let $A=\{a_{i,j}\}$ be the adjacency matrix of $\mathcal{N}'$. We associate the stochastic matrix $P=(p_{i,j})$, where $p_{i,j}=a_{i,j}/\deg'(i)$, with $\mathcal{N}'$. For each node i, define $\pi_i=\deg'(i)/(N+2M)$, where M is the number of edges of $\mathcal{N}$. Note that π_i can be computed by each processor i from knowledge it has of the topology of the network. We will prove the following theorem.

THEOREM 7. *Let $\mathcal{N}$ be an anonymous N-node network with maximal node valency d and let $\mathcal{N}'$ be the network $\mathcal{N}$ with self loops added to each node. Let ρ be the second largest eigenvalue (in absolute value) of the stochastic matrix P associated with $\mathcal{N}'$. There is an algorithm that computes any symmetric function on the network $\mathcal{N}$ with bit complexity*

$$O\left(\frac{-\log N}{\log \rho} \cdot N \cdot \log N \cdot d\right).$$

Proof. The idea of the algorithm is the following. Each processor sends its initial input value to all its neighbors. After receiving the values from all its neighbors the processor updates the value it already has based on "the average of the values" it receives. Every processor executes these steps S times, where S is of order $O(-\log N/\log \rho)$. Eventually in this way every input value to a node of the network will be distributed and equally accounted for by every other processor.

Let f be a symmetric Boolean function on N variables known to all the processors and let f_k be the value of f on inputs of weight k. Let

$I = \langle b_v : v \in V \rangle$ be the input to the network, where b_v is the input bit to node v. More formally the algorithm is as follows.

Algorithm for processor p:
Input: b_p, f;
Initialize: $value_p[0] := b_p$ and S;
for $i := 0, 1, ..., S$ **do**
  send $\dfrac{value_q[i]}{\deg'(p)}$ to all neighbors of p;
  receive $\dfrac{value_q[i]}{\deg'(q)}$ from all neighbors q of p;
  compute $value_p[i+1] :=$
  $\sum \left\{ \dfrac{value_q[i]}{\deg'(q)} : q \text{ is a neighbor of } p \text{ or } q = p \right\}$
od;
put $w := \left\lfloor \dfrac{value_p[S]}{\pi_p} \right\rfloor$;
Output f_w.

Each processor knows its input bit but does not know the network input configuration I. At the ith stage, $i \leqslant S$, processor p updates its variable $value_p$ which is an approximation to the number of 1's in the input configuration I times the quantity π_p. At the final stage the processor computes $w = [value_p/\pi_p]$, the nearest integer less than or equal to $value_p/\pi_p$. If the approximation is sufficiently close to the actual value $k\pi_p$, where k is the weight of the input I, then all processors will output the same correct value f_w.

We have to show that all the processors eventually converge to the correct ratio (and hence the resulting value f_w is the same for all the processors) and to bound the value of S. We will use the theory of Markov chains [21, 12] in order to complete the proof of correctness of the above algorithm.

Note that $P = (p_{i,j})$ is the $N \times N$ stochastic matrix of a primitive, reversible Markov chain corresponding to a random walk on $\mathcal{N}'$. (In general, the stochastic matrix corresponding to an arbitrary connected, undirected network is only irreducible and need not be primitive. By adding a self loop to each vertex of $\mathcal{N}$ to form the network $\mathcal{N}'$ we guarantee the corresponding Markov chain is primitive.) Its stationary distribution is $(\pi_1, \pi_2, ..., \pi_N)$. Let $1 = \lambda_1 > \lambda_2 \geqslant \cdots \geqslant \lambda_k$ be the eigenvalues of P and put

$$\rho = \max\{|\lambda_i| : 2 \leqslant i \leqslant k\}.$$

Standard arguments (see, e.g., [7, Lemma 2]) show that

$$p_{i,j}^{(r)} = \pi_j + O\left(\sqrt{\frac{\pi_j}{\pi_i}} \cdot \rho^r\right), \tag{1}$$

where $p_{i,j}^{(r)}$ is the (i, j) entry of the matrix P^r. If $M_P = \max_{i,j}\{\sqrt{\pi_j/\pi_i}\}$ then the matrix form of Eq. (1) is

$$P^r = P^\infty + O(M_P \cdot \rho^r \cdot E), \tag{2}$$

where E is the matrix of all 1's, and the limit $P^\infty = \lim_{r \to \infty} P^r$ of the chain is an $N \times N$ matrix such that all the entries of its ith column are equal to π_i. In our case, $M_P = \sqrt{d_{\max}/d_{\min}}$, where $d_{\max}$ (respectively, $d_{\min}$) is the maximal (respectively, minimal) valency of the network $\mathcal{N}$. Hence, Eq. (2) becomes

$$P^r = P^\infty + O\left(\sqrt{\frac{d_{\max}}{d_{\min}}} \cdot \rho^r \cdot E\right). \tag{3}$$

It is easy to see that for any input vector I, $L = IP^\infty$ is the eigenvector of P whose ith entry equals $k\pi_i$, where k is the number of 1's in the input I.

We are interested in the rate of convergence of the limit of IP^r as r tends to infinity. It follows from Eq. (3) that

$$IP^r = L + O\left(\sqrt{\frac{d_{\max}}{d_{\min}}} \cdot \rho^r \cdot k \cdot e\right) \tag{4}$$

where e is the row vector consisting of all 1's. During the rth iteration of the above algorithm processor p computes the pth component of IP^r. To guarantee that all the processors compute the correct value it is enough to ensure that the error term in (4) is less than $\frac{1}{2}\pi_p$; i.e.,

$$\sqrt{\frac{d_{\max}}{d_{\min}}} \cdot \rho^r \cdot k < \frac{1}{2(N+2M)}.$$

This inequality implies that the number S of iterations required is $S = O(-\log N/\log \rho)$, if $\rho > 0$. (Of course the case $\rho = 0$ is possible but then the number of required iterations is $S = 2$.) It is not hard to see that during each iteration of the algorithm $O(\log S)$ bits must be transmitted by each processor to all of its $\leqslant d$ neighbors in order to guarantee a sufficient precision of the approximation at the Sth iteration. By results of Landau and Odlyzko [15], for any network $\mathcal{N}$ with maximal node valency d

and diameter δ, the second largest eigenvalue of the stochastic matrix corresponding to the network $\mathcal{N}'$ satisfies the inequality

$$\rho \leqslant 1 - \frac{1}{N \cdot \delta \cdot (2+d)},$$

i.e., $\log \rho \leqslant -(1/N) \cdot \delta \cdot 2(2+d)$. Hence $\log S = O(\log N)$ and so the bit complexity of the algorithm (number of steps $\times$ number of processors $\times$ maximal number of bits per step per processor) is indeed

$$O\left(-\frac{\log N}{\log \rho} \cdot N \cdot \log N \cdot d\right),$$

as we had to prove. ∎

As an immediate consequence of the above theorem we get the following bound on the bit complexity of symmetric functions on anonymous networks.

THEOREM 8. *Let $\mathcal{N}$ be an anonymous N-node network with maximal node valency d and diameter δ. There is an algorithm that computes any symmetric Boolean function on $\mathcal{N}$ with bit complexity $O(N^2 \cdot \delta \cdot d^2 \log^2 N)$.*

Proof. As above $\log \rho \leqslant -1/N \cdot \delta \cdot (2+d)$. Combining this with Theorem 7 we obtain that the bit complexity for computing symmetric functions is $O(N^2 \cdot \delta \cdot d^2 \cdot \log^2 N)$. ∎

4.2. *Algorithms for Specific Networks*

Here are some applications of the theorem to specific types of networks.

COROLLARY 9. *The bit complexity of computing any symmetric function on an anonymous d-dimensional torus with $N = n^d$ nodes is $O(n^{1+2/d} \log^2 N)$.*

Proof. The characteristic values of the corresponding adjacency matrix of $\mathcal{N}'$ are given by the formula

$$1 + \sum_{k=1}^{d} 2 \cos\left(\frac{2\pi}{n} i_k\right), \qquad 1 \leqslant i_1, \ldots, i_d \leqslant n.$$

The second largest eigenvalue of the corresponding stochastic matrix of $\mathcal{N}'$ is

$$\rho = \frac{1}{2d+1} \cdot \left(1 + 2d \cdot \cos\left(\frac{2\pi}{n}\right)\right).$$

Using approximations to the log and cos functions it is easy to show $\log \rho = O(-1/n^2)$. Thus, by the theorem, the bit complexity of computing symmetric functions in this case is $O(N^{1+2/d} \log^2 N)$. ∎

COROLLARY 10. *The bit complexity of computing a symmetric function on an anonymous n-dimensional hypercube with* $N=2^n$ *nodes is* $O(N \log^4 N)$.

Proof. The eigenvalues of the adjacency matrix of the hypercube are $\lambda_i = n-2i$, $0 \leqslant i \leqslant n$. The second largest eigenvalue of the corresponding stochastic matrix of $\mathcal{N}'$ is $(n-1)/(n+1)$. Using the inequality $\log(1-2/(n+1)) < -2/(n+1)$, the theorem implies that the bit complexity of computing symmetric functions in this case is $O(N \log^4 N)$. ∎

For random regular graphs we can prove the following result.

COROLLARY 11. *The bit complexity of computing any symmetric function on a random regular graph of valency* $2d$ *is* $O(Nd \log^2 N/\log d)$ *with probability greater than* $1 - N^{-\Omega(\sqrt{d})}$.

Proof. This follows immediately from the theorem and recent results of Friedman *et al.* [11] bounding the size of the second largest eigenvalue of random regular graphs; i.e., with probability $1 - N^{\Omega(\sqrt{d})}$ the second largest eigenvalue of a regular graph of valency $2d$ is $O(\sqrt{d})$, where d is fixed as $N \to \infty$. ∎

5. DISTANCE REGULAR GRAPHS

In this section we show that by taking advantage of the topology of distance regular graphs we can derive efficient algorithms for computing symmetric functions on such graphs.

5.1. *Definitions and Examples*

The distance between any two nodes p, q of a network $\mathcal{N}$, denoted $d(p, q)$, is the length of the shortest path between p and q. The circle with center $p \in V$ and radius k, denoted by $C(p; k)$, is the set of nodes $q \in V$ such that $d(p, q) = k$. The set of neighbors of p, denoted $\mathcal{N}(p)$, is the circle $C(p; 1)$.

A network $\mathcal{N}$ is called distance regular if for any nodes p, q at distance k there are precisely a_k neighbors of p at distance $k-1$ from q and b_k neighbors of p at distance $k+1$ from q; i.e., the quantities

$$|C(p; 1) \cap C(q; k-1)|,$$

$$|C(p; 1) \cap C(q; k+1)|$$

do not depend on the particular nodes p, q but only on their distance $d(p, q)$.

Distance regular graphs were introduced by Biggs [6] and arose in the context of combinatorial regularity properties implied by the existence of an intersection array. Their theory has many connections to design theory, coding theory, geometry, and group theory [8].

Distance regular graphs include hypercubes, odd graphs, triangle graphs, complete bipartite graphs, etc. [6, 8, 9]. A network $\mathcal{N}$ is distance transitive if for any nodes p, q, p', q' with $d(p, q) = d(p', q')$, there is an automorphism ϕ of the network $\mathcal{N}$ such that $\phi(p) = p'$ and $\phi(q) = q'$. It is easy to see that all distance transitive graphs are distance regular, but the converse is false [6].

5.2. *Algorithm for Symmetric Functions*

Now we are ready to prove the main theorem of this section. First of all we remind the reader that the threshold function $\text{TH}_k \in B_N$ is defined to be 1 on inputs of weight at least k and 0 otherwise.

THEOREM 12. *On an anonymous N-node distance regular network of valency d and diameter δ every symmetric function can be computed in $O(N \cdot \delta \cdot d \cdot \log N)$ bits. Moreover the threshold function TH_k can be computed in $O(N \cdot \delta \cdot d \cdot \log k)$ bits, where $k \leqslant N$.*

Proof. For any processors p, q with $k = d(p, q)$ we define

$$a_k = |\{r \in C(p; 1) : d(q, r) = k - 1\}|, \qquad k = 1, 2, \ldots, \delta$$

$$b_k = |\{r \in C(p; 1) : d(q, r) = k + 1\}|, \qquad k = 0, 1, \ldots, \delta - 1,$$

$$c_k = |\{r \in C(p; 1) : d(q, r) = k\}|, \qquad k = 0, 1, \ldots, \delta.$$

(Note that $c_k = k - a_k - b_k$.) For any input configuration I, any processor p, and any distance $k \leqslant \delta$, let $I(p; k)$ be the number of processors at distance k from the processor p and whose input bit is 1. To compute a symmetric function it is sufficient for each processor p to know $I(p; k)$, for each $k \leqslant \delta$. The idea of the proof is to find an (inductive) formula for computing $I(p; k)$ in terms of the previously computed values $I(p; l)$, where $l < k$, and values $I(q, l)$, where $q \in C(p; 1)$ is a neighbor of p, $l < k$. We note that

$$\sum_{q \in \mathcal{N}(p)} I(q; k-1) = |\{\langle q, x\rangle : q \in \mathcal{N}(p),\ d(q, x) = k - 1,\ b_x = 1\}|$$

$$= \sum_{b_x = 1} |\{q \in \mathcal{N}(p) : d(q, x) = k - 1\}|$$

$$= \sum_{b_x = 1,\ d(p,\ x) = k} |\{q \in \mathcal{N}(p) : d(q, x) = k - 1\}|$$

$$+ \sum_{b_x = 1,\ d(p,\ x) = k - 1} |\{q \in \mathcal{N}(p) : d(q, x) = k - 1\}|$$

$$+ \sum_{b_x = 1,\ d(p,\ x) = k - 2} |\{q \in \mathcal{N}(p) : d(q, x) = k - 1\}|$$

$$= \sum_{\substack{b_x = 1 \\ d(p,\ x) = k}} a_k + \sum_{\substack{b_x = 1 \\ d(p,\ x) = k - 1}} c_{k-1} + \sum_{\substack{b_x = 1 \\ d(p,\ x) = k - 2}} b_{k-2}$$

$$= a_k \cdot I(p; k) + c_{k-1} \cdot I(p; k-1) + b_{k-2} \cdot I(p; k-2),$$

which in turn leads to the inductive formula

$$a_k I(p; k) = \sum_{q \in \mathcal{N}(p)} I(q; k-1) - (d - a_{k-1} - b_{k-1}) \cdot I(p; k-1) - b_{k-2} \cdot I(p; k-2). \tag{5}$$

Formula (5) and the knowledge of the network topology (i.e., the numbers a_k and b_k) make it possible to construct an efficient algorithm for computing symmetric functions. Let $f \in B_N$ be a symmetric function and let f_k be the value of f on inputs of weight k.

Algorithm for processor p:
Input: b_p, f;
Initialize: $I(p; 0) := 1$ if p's input bit is 1 and $:= 0$ otherwise;
send input bit to all neighbors;
compute $I(p; 1) :=$ the number of 1's among the neighbors of p;
for $k := 1, ..., \delta - 1$ **do**
    send $I(p; k)$ to all the neighbors of p;
    compute $I(p; k+1)$ from $I(p; k-1)$, $I(p; k)$ and the $I(q; k)$'s, where q ranges over all neighbors of p, via formula (5);
od;
compute the sum $s := \sum_{k=0}^{\delta} I(p; k)$;
output f_s

The correctness for the algorithm was shown above. It remains to determine its complexity. For $k = 0, ..., \delta$ each processor p transmits the number $I(p; k)$ to all its neighbors. This requires transmission of δ messages

$$I(p; 0), ..., I(p; \delta)$$

(each of length less than or equal to $\log N$ bits) to each of the d neighbors of p; i.e., $O(\delta \cdot d \cdot \log N)$ bits per processor for a total of $O(N \cdot \delta \cdot d \cdot \log N)$ bits.

The proof of the bit complexity of computing the threshold function TH_k employs the previous algorithm. Observe that when the number of 1's at a certain distance from a processor exceeds the threshold value k then we only need to transmit k, which requires $\log k$ bits. ∎

An important corollary to the above is the case of the hypercube.

COROLLARY 13. *On the anonymous hypercube, every symmetric function can be computed in* $O(N \cdot \log^3 N)$ *bits. Moreover the threshold function* TH_k *can be computed in* $O(N \cdot \log^2 N \cdot \log k)$ *bits, where* $k \leqslant N$.

Proof. Let $n = \log N$. This is an immediate consequence of the fact that the hypercube is distance regular. It is easy to show that in the notation of section 5, $a_k = k$, $b_k = n - k$, and $c_k = 0$. The resulting inductive formula (which is a special case of formula (5)) is the following:

$$b(p; k) = \frac{1}{k} \cdot \left(\sum_{q \in D(p;1)} b(q; k-1) - (n-k+2) \cdot b(p; k-2) \right). \qquad (6)$$

This proves our assertion. ∎

6. CONCLUSIONS AND OPEN PROBLEMS

The present paper has been concerned with the problem of determining algorithms with polynomial bit complexity for computing Boolean functions on anonymous distributed networks. The main result of Section 3 provides such an algorithm for any anonymous network $\mathcal{N}$ with bit complexity $O(N^3 \cdot \delta \cdot d^2 \cdot \log N)$. It would be interesting, however, if we could improve on this bit complexity.

We have been able to find more efficient algorithms for computing symmetric functions on arbitrary networks (Theorem 8) and very efficient algorithms for symmetric functions on the class of distance regular networks (Theorem 12). Nevertheless, these algorithms are still not known to be optimal and improvements are possible.

An interesting special case is that of the hypercube network. Based upon the results of [4] for unlabeled and oriented rings and [5] for oriented tori one would expect that there are more efficient algorithms for computing Boolean functions on the unlabeled and oriented hypercube than those provided here. For the oriented hypercube this is indeed confirmed by results in [14], which gives an algorithm computing all computable Boolean functions (not just symmetric) with bit complexity $O(N \log^4 N)$. Nevertheless the unoriented hypercube remains a mystery and nothing better is known than the $O(N^3 \log^4 N)$ result implied by Theorem 6.

There have been few studies in the literature regarding lower bounds on the bit complexity. The only network for which this question has been studied extensively is the ring [1, 10, 17]. Reference [19] studies the question for the extrema finding function but relies on specific properties of this function. Reference [24] give lower bounds for the message complexity of computing Boolean functions for broad classes of networks. However, very little is known about lower bounds on the bit complexity of Boolean functions on the anonymous torus or hypercube, not to mention the general case of unlabeled networks.

If we allow the processors to flip coins in the course of the computation then this changes entirely the rules of the game. It is now possible to introduce algorithms with improved average and worst case bit complexity. Also, the class of functions computable in this model may be different. For the case of rings this has been studied by [3]. For general networks [20] and [16] have given algorithms with low message complexity for the problem of constructing a rooted spanning tree (which can then be used to compute Boolean functions efficiently). It would be very interesting to examine more thoroughly the bit complexity for the case of general anonymous networks.

ACKNOWLEDGMENTS

We are grateful to L. Meertens, J. Hastad and J. Tromp for many fruitful conversations. H. Attiya and T. Tsantilas were very helpful with the bibliography. Also, many thanks to M. Merritt and the anonymous referee for their careful reading of the paper and for making numerous suggestions that significantly improved the presentation.

RECEIVED July 25, 1990; FINAL MANUSCRIPT RECEIVED April 24, 1992

REFERENCES

1. ABRAHAMSON, K., ADLER, A., HIGHAM, L., AND KIRKPATRICK, D. (1988), Randomized evaluation on a ring, *in* "Distributed Algorithms, 2nd International Workshop, Amsterdam, The Netherlands, July 1987" (J. van Leeuwen, Ed.), pp. 324–331, Lecture Notes in Computer Science, Vol. 312, Springer-Verlag, Berlin/New York.
2. ANGLUIN, D. (1980), Local and global properties in networks of processors, *in* "12th Annual ACM Symposium on Theory of Computing," pp. 82–93.
3. ATTIYA, H., AND SNIR, M. (1991), Better computing on the anonymous ring, *J. Algorithms* **12**, 204–238.
4. ATTIYA, H., SNIR, M., AND WARMUTH, M. (1988), Computing on an anonymous ring, *J. Assoc. Comput. Mach.* **35**, 845–875. A short version appeared *in* "Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computation," (1985).
5. BEAME, P.W., AND BODLEANDER, H. L. (1989), Distributed computing on transitive networks: The torus, *in* "6th Annual Symposium on Theoretical Aspects of Computer Science, STACS" (B. Monien and R. Cori, Eds.), pp. 294–303, Lecture Notes in Computer Science, Springer-Verlag, Berlin/New York.

6. BIGGS, N. (1974), "Algebraic Graph Theory," Cambridge Univ. Press, London/New York.
7. BRODER, A., AND KARLIN, A. (1989), Bounds on the cover time, *J. Theoret. Probab.* **2**(1), 101–120.
8. BROUWER, A. E., COHEN, A. M., AND NEUMAIER, A. (1989), "Distance-Regular Graphs," Ergebnisse der Mathematik und ihrer Grenzgebiete, 3. Folge, Vol. 18, Springer-Verlag, Berlin/New York.
9. CAMERON, P. J. (1983), Automorphism groups of graphs, *in* "Selected Topics in Graph Theory" (L. W. Beineke and R. J. Wilson, Eds.), Vol. 2, Chap. 4, pp. 89–127, Academic Press, San Diego.
10. DURIS, P., AND GALIL, Z. (1987), Two lower bounds in asynchronous distributed computation, *in* "Proceedings 28th Annual IEEE Symposium on Foundations of Computer Science," pp. 326–330.
11. FRIEDMAN, J., KAHN, J., AND SZEMERÉDI, E. (1989), On the second eigenvalue of random regular graphs, *in* "21st Annual ACM Symposium on Theory of Computing," pp. 587–598.
12. GANTMACHER, F. R. (1959), "Matrix Theory," Chelsea New York. [Translated from the Russian]
13. KRANAKIS, E., AND KRIZANC, D. (1990), "Computing Boolean Functions on Cayley Networks," Technical Report CS R90-61, Centrum voor Wiskunde en Informatica, Department of Algorithms and Architecture.
14. KRANAKIS, E., AND KRIZANC, D. (1991), Distributed computing on anonymous hypercube networks, *in* "Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas."
15. LANDAU, H. J., AND ODLYZKO, A. M. (1981), Bounds for eigenvalues of certain stochastic matrices, *Linear Algebra Appl.* **38**, 5–15.
16. MATIAS, Y., AND AFEK, Y. (1989), Simple and efficient election algorithms for anonymous networks, *in* "Distributed Algorithms, 3nd International Workshop, Nice, France" (J. C. Bermond and M. Raynal, Eds.), pp. 183–194, Lecture Notes in Computer Science, Vol. 392, Springer-Verlag, Berlin/New York.
17. MORAN, S., AND WARMUTH, M. (1986), Gap theorems for distributed computation, *in* "5th Annual ACM Symposium on Principles of Distributed Computation," pp. 131–140.
18. NORRIS, N. (1990), "Universal Covers off Edge-Labeled Digraphs: Isomorphism to Depth $n-1$ Implies Isomorphism to All Depths," Technical Report UCSC-CRL-90-49, Univ. of California at Santa Cruz.
19. PACHL, J., KORACH, E., AND ROTEM, D. (1984), A new technique for proving lower bounds for distributed maximum finding algorithms, *J. Assoc. Comput. Mach.* **31** (4), 905–918.
20. SCHIEBER, B., AND SNIR, M. (1989), Calling names on nameless networks, *in* "8th Annual ACM Symposium on Principles of Distributed Computation," pp. 319–328.
21. SENETA, E. (1981), "Non-negative Matrices and Markov Chains," 2nd ed., Springer Series in Statistics, Springer-Verlag, Berlin/New York.
22. TEL, G. (1988), Directed network protocols, *in* "Proceedings of 2nd International Workshop on Distributed Algorithms, Amsterdam" (J. van Leeuwen, Ed.), pp. 13–29, Lecture Notes in Computer Science, Vol. 312, Springer-Verlag, Heidelberg.
23. YAMASHITA, M., AND KAMEDA, T. (1987), "Computing Functions on an Anonymous Network," Technical Report 87-16, Laboratory for Computer and Communication Research, Simon Fraser University.
24. YAMASHITA, M., AND KAMEDA, T. (1988), Computing on an anonymous network, *in* "7th Annual ACM Symposium on Principles of Distributed Computation," pp. 117–130.