Centrum voor Wiskunde en Informatica

Centre for Mathematics and Computer Science

E. Kranakis, D.D.M. Krizanc

Computing Boolean functions on anonymous networks

Computer Science/Department of Algorithmics & Architecture Report CS-R8935 September

1989

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

E. Kranakis, D.D.M. Krizanc

Computing Boolean functions on anonymous networks

Computer Science/Department of Algorithmics & Architecture Report CS-R8935 September

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

COMPUTING BOOLEAN FUNCTIONS ON ANONYMOUS NETWORKS

Evangelos Kranakis [1]
(eva@cwi.nl)

Danny Krizanc [1,2]
(krizanc@cs.rochester.edu)

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

(2) University of Rochester, Department of Computer Science
Rochester, New York, 14627, USA

Abstract

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, unlabeled networks we give a general algorithm of polynomial bit complexity $O(N^4 \cdot \delta \cdot d^2 \cdot \log N)$ for computing any boolean function which is computable in this network. This improves the previous best known algorithm which was of exponential bit complexity $O(d^{N^2})$. We consider the class of distance regular unlabeled networks and show that in such networks symmetric functions can be computed efficiently in $O(N \cdot \delta \cdot d \cdot \log N)$ bits. This compares favorably with a lower bound $\Omega(N \cdot \delta \cdot d)$ bits for symmetric functions on regular networks. We also consider the n-dimensional hypercube, with $N = 2^n$ nodes. We show that in the oriented hypercube an arbitrary boolean function f is computable if and only if it is kept invariant under all the flipping automorphisms of the hypercube, in which case it can be computed in $O(N^2)$ bits. Further we show that every symmetric function can be computed in $O(N \cdot \log^2 N)$ bits on the oriented and in $O(N \cdot \log^3 N)$ bits on the unlabeled hypercube.

1980 Mathematics Subject Classification: 68Q99
CR Categories: C.2.1
Key Words and Phrases: Anonymous network, boolean function, distance regular graph, distance transitive graph, group of automorphisms, hypercube, labeled and unlabeled networks, oriented and unoriented networks, ring, symmetric boolean function, threshold function, torus, transitive graph.
Note: This paper will submitted for publication elsewhere.

Report CS-R8935
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1 Introduction

A very important problem in distributed computing is the designing of efficient algorithms for computing boolean functions in distributed networks of processors. For both practical and theoretical considerations it is useful to minimize the total number of exchanged bits which are necessary in order to compute a certain boolean function, but at the same time keeping the processors as similar to each other as possible.

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, either 0 or 1.

The processors follow a deterministic protocol (or algorithm). During each step of the protocol they perform certain computations depending on their input value, their previous history and the messages they receive from their neighbors and then 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 certain bit. Let B_N be the set of boolean functions on N variables. Let $\mathcal{N} = (V, E)$ be a network of size N, with node set $V = \{0, 1, \ldots, N-1\}$ and edge set $E \subseteq V \times V$. An input to $\mathcal{N}$ is an N-tuple $I =< b_v : v \in V >$ 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 $I =< b_v : v \in V >$ each processor $v \in V$ starting with the input bit b_v should terminate its computation according to the given protocol and output the value b such that $f(I) = b$. A network computes the function f if for each input I, at the end of the computation each processor computes correctly 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 boolean functions.

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 are anonymous (this means that they do not know either the identities of themselves or of the other processors),
3. the processors are identical (this means they all run the same algorithm),
4. the processors are deterministic,
5. the network is asynchronous,
6. the network may or may not be oriented (by orientation we mean a global, consistent labeling of the network links).

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 [ASW85]. Angluin [Ang80] has shown that if the processors are anonymous and identical there is no algorithm for electing a leader. 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. The last condition on orientation will be discussed in the next section.

1.1 Labeled versus Unlabeled Networks

Before proceeding with an outline of the main results of the paper it will be useful to clarify the notions of labeled, unlabeled and oriented networks and their impact on computability questions. By a labeling of the network $\mathcal{N} = (V, E)$ we understand a function that for all nodes $v \in V$, with degree $deg(v)$, associates the values $1, 2, \ldots, deg(v)$ to the links incident with v. More formally it is a function, $\mathcal{L}$, on the set $\{(x, y), (y, x) : \{x, y\} \in E\}$, such that for each node $v \in V$ the mapping $u \longrightarrow \mathcal{L}(v, u)$ is $1 - 1$ on the set of neighbors u of v. Note that in general $\mathcal{L}(u, v) \neq \mathcal{L}(v, u)$. If a network $\mathcal{N}$ has an associated labeling $\mathcal{L}$ then it is called a labeled network and is usually denoted by $\mathcal{N}[\mathcal{L}]$. Otherwise it is called an unlabeled network. If we want to emphasize that a certain labeling is known to all processors of the network then we call the labeling an orientation. Of special interest are the canonical orientations of the following three networks: rings, tori, and hypercubes.

The ring R_N consists of N processors arranged in a ring in such a way that processors i, j are adjacent if and only if $j = i \pm 1 \bmod N$. For the ring R_N we define an orientation as follows: $\mathcal{L}(i, i+1) = 1$ and $\mathcal{L}(i+1, i) = 2$, where addition is modulo N. The two dimensional $n \times n$ torus, with $N = n^2$ nodes, is the standard two dimensional mesh with wrap-around edges and side consisting of n nodes. We define a labeling of the torus as follows: the edges of node (x, y) corresponding to $(x, y + 1)$, $(x + 1, y)$, $(x, y - 1)$, $(x - 1, y)$ (where addition is modulo n) are labeled 1 (up), 2 (right), 3 (down), 4 (left), respectively. The oriented hypercube will be defined in section 4.

Let $Aut(\mathcal{N})$ be the group of automorphisms of the network $\mathcal{N}$. It is clear that $Aut(\mathcal{N})$ is a subgroup of the symmetric group of permutations S_N. A boolean function $f \in B_N$ is invariant under a permutation $\sigma \in S_N$ if for all inputs $x_1, \ldots, x_N$, $f(x_1, \ldots, x_N) = f(x_{\sigma(1)}, \ldots, x_{\sigma(N)})$. The automorphism group of a network provides a necessary (but not always sufficient) condition for a boolean function to be computable on the network. It is easy to show that any boolean function $f \in B_N$ computable on a network $\mathcal{N}$ is invariant under all the automorphisms of the network. This result will be very useful in distinguishing between oriented and unlabeled networks. An automorphism ϕ of the network $\mathcal{N}$ is consistent with a labeling $\mathcal{L}$ if for any adjacent nodes x, y, $\mathcal{L}(x, y) = \mathcal{L}(\phi(x), \phi(y))$. A labeling of the edges of $\mathcal{N}$ is consistent with a group $G \leq Aut(\mathcal{N})$ of automorphisms of $\mathcal{N}$ if any automorphism $\phi \in G$ is consistent with $\mathcal{L}$. We denote by $Aut(\mathcal{N}[\mathcal{L}])$ the group of automorphisms of $\mathcal{N}$ that are consistent with $\mathcal{L}$. In the same manner we can show that any function computable on the network $\mathcal{N}[\mathcal{L}]$ is invariant under the group of automorphisms $Aut(\mathcal{N}[\mathcal{L}])$. It is not hard to show that oriented networks are more powerful than unlabeled ones. As a matter of fact we can prove the following result.

Theorem 1.1 *For $N \geq 6$, there is a boolean function $f \in B_N$ computable on the oriented ring (torus, hypercube) but not computable on the unlabeled ring (torus, hypercube).* □

The proof of the theorem is not difficult. One way to prove it is by providing a group theoretic characterization of the boolean functions computable on the corresponding oriented network. Such a characterization is in fact possible for all the oriented networks listed above. The group of permutations of S_N that leave the boolean function f invariant is called invariance group of f (see [CK89]) and is denoted by $S(f)$. It follows from the main

result of [ASW85] that a boolean function $f \in B_N$ is computable on the oriented ring R_N if and only if it is invariant under the cyclic group C_N (this is the group of automorphisms generated by the N-cycle $(0, 1, \ldots, N-1)$) while it is computable on the unlabeled ring R_N if and only if it is invariant under the dihedral group D_N (this is the group of automorphisms generated by the N-cycle $(0, 1, \ldots, N-1)$ and the reflection permutation). Similar considerations show that the boolean functions $f \in B_N$ computable on the 2 dimensional, $n \times n$, oriented torus, are exactly the ones which are invariant under the group $C_n \otimes C_n$. The case of the hypercube will be handled separately in theorem 4.3.

Thus to every network $\mathcal{N}$, with labeling $\mathcal{L}$, there corresponds the class of functions computable on $\mathcal{N}[\mathcal{L}]$ and denoted by $\mathcal{F}^{\mathcal{N}[\mathcal{L}]}$. The above observations clearly indicate that this class of functions depends on the labeling of the network considered. However, if $\mathcal{F}^{\mathcal{N}}$ is the class of functions computable on the unlabeled network $\mathcal{N}$ then it is easy to show that

$$\mathcal{F}^{\mathcal{N}} = \bigcap_{\mathcal{L}} \mathcal{F}^{\mathcal{N}[\mathcal{L}]},$$

where $\mathcal{L}$ above ranges over all labelings of $\mathcal{N}$. The main goal of the present paper is to provide efficient algorithms for computing the class of functions $\mathcal{F}^{\mathcal{N}[\mathcal{L}]}$ for labeled and $\mathcal{F}^{\mathcal{N}}$ for unlabeled networks.

1.2 Outline and Results of the Paper

In the sequel we assume that N is the number of processors in a given anonymous network. The simplest topology considered in the study of the bit complexity of computing boolean functions is the ring e.g., [AAHK88], [ASW85], [AS88], [MW86], [PKR84]. It has been shown by [ASW85] that there is an algorithm for computing all (computable on the ring) boolean functions with bit complexity $O(N^2)$. Moreover, this bit complexity is the same on both oriented and unlabeled rings. In addition, [MW86] 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 oriented torus [BB89] give an algorithm with bit complexity $O(N^{1.5})$, and construct nonconstant functions with bit complexity $\Theta(N)$. For general graphs [YK87a] and [YK87b] show that the message complexity of computing a boolean function on an arbitrary unlabeled 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 and fanout the corresponding 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).

In the present paper we study the bit complexity for boolean functions on arbitrary unlabeled networks and on distance regular networks. We show in section 2 that for any unlabeled N-node network of maximal node valency d and diameter δ, every boolean function which is computable on the network can be computed in $O(N^4 \cdot \delta \cdot d^2 \cdot \log N)$ bits, thus significantly improving the previous $O(d^{N^2})$ upper bound of [YK87b]. For the case of distance regular networks we show in section 3 how to compute any symmetric function in $O(N \cdot \delta \cdot d \cdot \log N)$ bits. Since for symmetric functions on regular graphs we can prove an optimal $\Omega(N \cdot \delta \cdot d)$ lower bound on the bit complexity, our algorithm in this case is within a $\log N$ factor of optimal. In section 4 we give a characterization of the functions computable on the oriented hypercube, as the boolean functions which are kept invariant under the group of automorphisms of the oriented hypercube and provide an $O(N^2)$ upper bound for general computable boolean functions, and an $O(N \cdot \log^2 N)$ upper bound for

symmetric boolean functions. For unlabeled hypercubes we give an $O(N \cdot \log^3 N)$ upper bound for symmetric boolean functions. We conclude in section 5 with some discussion and open problems.

2 Unlabeled Networks

In this section we give a general algorithm which computes any boolean function computable on a given network using polynomial bit complexity. One of the results that will be used very frequently in the sequel concerns the computation of certain simple operations, like maximum and set-union on general unlabeled 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. $\Diamond : A \times A \longrightarrow A$ 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 unlabeled network and let $\Diamond$ be an operation satisfying the above three conditions. Let A^N be the set of all N-tuples from elements of A. For any input $I =< i_p : p \in V >\in A^N$ to the network we can define a function $\Diamond : A^N \longrightarrow A$ 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 $\Diamond : A \times A \longrightarrow A$ and the function $\Diamond : A^N \longrightarrow A$.) In view of the associativity of $\Diamond$ this function is well defined. As a first step in our goal for 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.

Theorem 2.1 *Let $\mathcal{N}$ be an unlabeled 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 $I =< i_p : p \in N >\in A^N$ 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 $I =< i_p : p \in V >$ be the input to the network.

Algorithm for procesor p:
Initialize: $value_p[0] := i_p$;
for $i := 0, 1, \ldots, \delta - 1$ **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. By commutativity and associativity it is immaterial the order in which the operation $\Diamond$ 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. This is worth stating separately.

Corollary 2.1 *On an unlabeled N-node network with maximal node valency d and diameter δ the OR_N function can be computed with bit complexity $O(N \cdot \delta \cdot d)$.*

Proof. Apply theorem 2.1 to the operation of binary or, i.e. $a \Diamond b = a \vee b$. □
If the network is regular then OR_N requires $\Omega(N \cdot \delta \cdot d)$ bits. For a proof of this see [ASW85] or theorem 3.2. Thus for this case the above algorithm is optimal.

Another corollary will be useful in the proof of our general theorem 2.2 about the bit complexity of computable boolean functions on general networks.

Corollary 2.2 *Let $\mathcal{N}$ be an unlabeled N-node network with maximal node valency d and diameter δ. There is an algorithm for computing the set $\{i_p : p \in N\}$ for any input $I = < i_p : p \in V > \in A^N$ 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 2.1 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 $\{i_p : p \in N\}$. Each element can be coded with α bits, and therefore such sets can be coded with $N \cdot \alpha$ bits. □

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

Theorem 2.2 *Let $\mathcal{N}(V, E)$ be an unlabeled 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 $O(N^4 \cdot \delta \cdot d^2 \cdot \log N)$.*

Proof. Our algorithm relies on several cost efficient adjustments and improvements of the algorithm of [YK87a] using Theorem 2.1. Let $f \in B_N$ be any computable boolean function on the anonymous network $\mathcal{N}$. Let $I =< b_p : p \in V >$ be the input to the network, where b_p is the input to node p. We present the algorithm in three phases.

Phase 1. Each processor chooses an arbitrary labeling for all its incident edges, i.e., the links of p are labeled with the numbers $1, 2, \ldots, deg(p)$, where $deg(p)$ is the degree of p. Now each processor transmits to each neighbor the label it has chosen for the link connecting them. Let $\mathcal{L}$ be the resulting labeling of the network $\mathcal{N}$. Next, each pair (p, q) of processors labels their corresponding link with

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

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 2. In this phase each processor gathers as much information as possible 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, $T_{\mathcal{L},I}(p)$ [YK87b]. Since $\mathcal{L}$ 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^2. 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 the node 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.

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 have exponential bit complexity $\Omega(d^i)$ and transmitting them is rather expensive. Therefore we must be careful if we want to achieve an algorithm with polynomial bit complexity. 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, $i = 0, 1, \ldots, N^2$, by executing the following algorithm.

Algorithm for procesor p:
Initialize: $T_p^0 := b_p$ **and** $set_p^0 := \{T_p^0\}$;
for $i := 0, \ldots, N^2$ **do**
  compute the set $set_p^i := \{T_q^i : q \in V\}$;
  code the elements of the set set_p^i with integers $1, \ldots, k$, where $k \leq N$ is the number of elements of 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 $code(T_q^i)$, where q is a neighbor of p;
  send the tree T_p^{i+1} to all the neighbors of p;
od;
output $set_p^{N^2}$.

After the trees of level i have been constructed the processors use the set algorithm given in corollary 2.2 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. The processors themselves can decode the trees in order to 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}. 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. As indicated above we iterate this algorithm N^2 times.

Phase 3. At this point all processors have computed the set of all views of depth N^2, namely the set $\{T_p^{N^2} : p \in V\}$. As in [YK87b] 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 3.3 in [YK87b] for any two trees if their restrictions to depth N^2 are isomorphic then the full trees themselves must also be isomorphic. Let $[T]_{I,\mathcal{L}}$ 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. Further, it follows from theorem 4.1 in [YK87b] 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 $\mathcal{L}, \mathcal{L}'$, if $[T]_{I,\mathcal{L}} = [T']_{I',\mathcal{L}'}$, for any trees T, T', then $f(I) = f(I')$. The processors want to compute $f(I)$, but they do not know the input I. To resolve this problem the processor uses its knowledge of the network topology to construct a labeling $\mathcal{L}'$ and an input I' such that $[T]_{I,\mathcal{L}} = [T]_{I',\mathcal{L}'}$, for all trees T. Certainly, each processor may choose a different input I' and labeling $\mathcal{L}'$. However by exchanging information using corollary 2.2 the processors can agree on a unique input I' and labeling $\mathcal{L}'$. Since the value of f 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. Phases 1 and 3 either involve local computations which do not require any bit exchanges or simple low cost bit exchanges. The main bit exchanges take place in phase 2. There we have N^2 iterations of the algorithm in corollary 2.2. We need $d \cdot \log N$ bits to represent each of the corresponding trees. This means that the bit complexity of the algorithm is $O(N^4 \cdot \delta \cdot d^2 \cdot \log N)$. □

As an application of the main theorem we conclude that every boolean function which is computable on the unlabeled torus (respectively, hypercube) can be computed with $O(N^{4.5} \cdot \log N)$ (respectively, $O(N^4 \cdot \log^4 N)$) bits.

3 Symmetric Functions in 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.

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

$(d(p,q) \leq k)$. The set of neighbors of p, denoted $\mathcal{N}(p)$, is the circle $C(p;1)$. The threshold function $Th_k \in B_N$ is defined to be 1 on inputs of weight at least k and 0 otherwise. (By the weight of an input I we understand the number of occurrences of 1 in the input.)

Distance regular graphs are graphs $\mathcal{N}$ such that for any nodes $p, q \in V$ with $d(p,q) = k$ the quantities

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

depend only on the distance $d(p,q)$. More formally, for $k = d(p,q)$ we define

$$\begin{aligned} a_k &= | \{r \in C(p;1) : d(q,r) = k-1\} |, k = 1,2,\ldots,\delta \\ b_k &= | \{r \in C(p;1) : d(q,r) = k+1\} |, k = 0,1,\ldots,\delta-1, \\ c_k &= | \{r \in C(p;1) : d(q,r) = k\} |, k = 0,1,\ldots,\delta. \end{aligned}$$

Such graphs include hypercubes, odd graphs, triangle graphs, complete bipartite graphs, etc. [Big74], [Cam83]. They satisfy several useful properties. We mention only a few obvious ones and refer the reader to [Big74] and [Cam83] for further properties. Distance regular graphs are regular with valency $d = b_0$. By definition, $a_0 = 0$. Moreover, $c_0 = 0$ and $a_1 = 1$. Since, if $d(p,q) = k$ every neighbor of p has distance k, $k-1$ or $k+1$ from q it is clear that $c_k = d - a_k - b_k$. 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 $\phi \in Aut(\mathcal{N})$ 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 [Big74].

Now we are ready to prove the main theorem of this section.

Theorem 3.1 *On an unlabeled 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 $k \leq N$.*

Proof. For any input configuration $I =< b_v : v \in V >$, any processor p and any distance $k \leq \delta$ let $I(p;k)$ be the number of processors x at distance k from the processor p such that $b_x = 1$. To compute a symmetric function it is sufficient for each processor p to know $I(p;k)$, for each $k \leq \delta$. The idea of the proof is to find a (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

$$\begin{aligned} \sum_{q \in \mathcal{N}(p)} I(q;k-1) &= | \{< q,x >: q \in \mathcal{N}(p), d(q,x) = k-1, b_x = 1\} | \\ &= \sum_{b_x=1} | \{q \in \mathcal{N}(p) : d(q,x) = k-1\} | \\ &= \sum_{b_x=1, d(p,x)=k} | \{q \in \mathcal{N}(p) : d(q,x) = k-1\} | + \\ &\quad \sum_{b_x=1, d(p,x)=k-1} | \{q \in \mathcal{N}(p) : d(q,x) = k-1\} | + \\ &\quad \sum_{b_x=1, d(p,x)=k-2} | \{q \in \mathcal{N}(p) : d(q,x) = k-1\} | \\ &= \sum_{b_x=1, d(p,x)=k} a_k + \sum_{b_x=1, d(p,x)=k-1} c_{k-1} + \sum_{b_x=1, d(p,x)=k-2} b_{k-2} \\ &= a_k \cdot I(p;k) + c_{k-1} \cdot I(p;k-1) + b_{k-2} \cdot I(p;k-2), \end{aligned}$$

which in turn leads to the following inductive formula

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

Formula (1) 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:
initialize: $I(p;0) := 1$ if p's input bit is 1 and is $:= 0$ otherwise;
send input bit to all neighbors;
compute $I(p;1) :=$ the number of 1s among the neighbors of p;
for $k := 1, \ldots, \delta - 1$ **do**
    send $I(p;k)$ to all the neigbors 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 (1);
od;
compute the sum $s := \sum_{k=0}^{\delta} I(p;k)$;
output f_s

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

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

(each of length $\leq \log N$ 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 $O(N \cdot \delta \cdot d \cdot \log N)$.

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

As mentioned above every distance transitive graph is distance regular. Distance transitive graphs include the complete graphs K_N, the complete bipartite graphs $K_{n,n}$, with $N = 2 \cdot n$, the rings R_N, the hypercubes Q_n with $N = 2^n$ (which will be studied in detail in section 4), the odd graphs O_k, $k \geq 2$, and the graphs $J(n,m)$. For more examples we refer the reader to [Big74] and [Cam83] and the references thereof.

We conclude this section with a lower bound on the bit complexity of computing symmetric functions on regular networks. It shows that the result of theorem 3.1 is optimal up to a factor $\log N$.

Theorem 3.2 *On an unlabeled regular network of valency d and diameter δ the bit complexity of every nonconstant symmetric function is $\Omega(N \cdot \delta \cdot d)$.*

Proof. The proof is similar to the proof of theorem 4.1 in [ASW85]. Let A be an algorithm computing a nonconstant symmetric function f. There exists an integer k such that for all inputs I, I' of weights k, $k+1$, respectively, $f(I) \neq f(I')$. We show that execution of the algorithm with input I will require $\Omega(N \cdot \delta \cdot d)$ bits. Say the algorithm accepts I after exactly

t steps. Since the network is unlabeled, at each time step at least d bits are transmitted by each processor to all its neighbors. Thus the processors terminate after sending at least $N \cdot (t-1) \cdot d$ bits. To prove the theorem it is sufficient to show that $t = \Omega(\delta)$. There exist processors p, q such that $d(p,q) = \delta$. Suppose that I and I' are inputs as above which differ only on their input bit at processor q. We claim that $\delta \leq t$. Assume to the contrary that $t < \delta$. Execute the algorithm A with input I'. After $\delta - 1$ steps the processor p will be in exactly the same state as it was when the input was I and must therefore output the same value as before. But this is a contradiction. □

4 Boolean Functions in Hypercubes

In this section we study the bit complexity of boolean functions in unlabeled and oriented hypercubes.

4.1 Unlabeled Hypercubes

For general boolean functions we have the following immediate consequence of Theorem 2.2.

Theorem 4.1 *On the unlabeled hypercube Q_n, $N = 2^n$, any boolean function which is computable on the network can be computed with bit complexity $O(N^4 \cdot \log^4 N)$.* □

Regarding symmetric functions on the unlabeled hypercube we have the following result which is an immediate consequence of Theorem 3.1.

Theorem 4.2 *On the unlabeled 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 $k \leq N$.*

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 3, $a_k = k$, $b_k = n - k$ and $c_k = 0$. The resulting inductive formula (which is a special case of formula (1)) is the following:

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

4.2 Oriented Hypercubes

In this section we study the bit complexity of computing boolean functions on oriented hypercubes and provide an algorithm with bit complexity $O(N^2)$ for computing such functions.

The nodes of the n dimensional hypercube Q_n consist of all sequences of bits $(x_1, \ldots, x_n)$ of length n. Two such nodes are adjacent when they differ in exactly one component. We distinguish two types of automorphisms of the hypercube: (1) the **flipping** automorphisms that flip the bits of certain components, i.e. for any set $S \subseteq \{1, \ldots, n\}$ let $\phi_S(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, where $y_i = x_i + 1$, if $i \in S$, and $y_i = x_i$ otherwise (here addition is modulo 2); and (2) the **permuting** automorphisms that permute the components according to

a fixed permutation of $\{1,\ldots,n\}$, i.e. for any permutation $\sigma \in S_n$, $\phi_\sigma(x_1,\ldots,x_n) = (x_{\sigma(1)},\ldots,x_{\sigma(n)})$. Notice that for any sets $S,T \subseteq \{1,\ldots,n\}$ and any index i, we have that $\phi_S \circ \phi_T = \phi_{S\Delta T}, \phi_{\{i\}} = \phi_{(1,i)} \circ \phi_{\{1\}} \circ \phi_{(1,i)}$, where $S \Delta T$ is the symmetric difference of S,T. Let F_n denote the group of flipping automorphisms and P_n the group of permuting automorphisms of Q_n.

A natural orientation of the hypercube is the following labeling $\mathcal{L}$: the edge connecting nodes $x = (x_1,\ldots,x_n)$ and $y = (y_1,\ldots,y_n)$ is labeled by i if and only if $x_i \neq y_i$, i.e. $\mathcal{L}(x,y) = \mathcal{L}(y,x) = i$. It is easy to see that this labeling is consistent for the group of flipping automorphisms, in the sense that if nodes x and y are labeled by i then so are $\phi(x)$ and $\phi(y)$, for any flipping automorphism ϕ.

Lemma 4.1 *F_n is a normal subgroup of $Aut(Q_n)$. Every automorphism ϕ of the unoriented hypercube is of the form $\phi_S \circ \phi_\sigma$ for some flipping automorphism ϕ_S and some permuting automorphism ϕ_σ. If C_2 is the permutation group on 2 elements generated by the cycle $(1,2)$ then $F_n = C_2 \otimes \cdots \otimes C_2$, n times, and $Aut(Q_n) = F_n \cdot P_n$. Moreover, the group of automorphisms of the oriented hypercube $Q_n[\mathcal{L}]$ is exactly the group F_n of flipping automorphisms.*

Proof. The lemma is not difficult to prove using the fact that Q_n is the graph product of n rings R_2. Normality of F_n follows from the identity $\phi_\sigma^{-1} \circ \phi_S \circ \phi_\sigma = \phi_{\sigma^{-1}(S)}$. On the other hand it is easy to see no permuting automorphism can be consistent with the above labeling. It follows that $Aut(Q_n[\mathcal{L}])$ is exactly the group of flipping automorphisms. □

We can prove the following theorem for this natural orientation.

Theorem 4.3 *On the oriented hypercube Q_n of degree n and for any boolean function $f \in B_N$, $N = 2^n$, f is computable on the hypercube Q_n if and only if f is invariant under the flipping automorphisms of Q_n. Moreover, the bit complexity of any such computable function is $O(N^2)$.*

Proof. The if part is easy. We need only prove the only if part. Let $f \in B_N$ be invariant under all flipping automorphisms of the hypercube. The algorithm proceeds by induction on the dimension n of the hypercube. Intuitively, it splits the hypercube into two $n-1$ dimensional hypercubes. The first hypercube consists of all nodes with $x_n = 0$ and the second of all nodes with $x_n = 1$. By the induction hypothesis the nodes of these hypercubes know the entire input configuration of their corresponding hypercubes. Every node in the hypercube with $x_n = 0$ is adjacent to unique node in the hypercube with $x_n = 1$. By exchanging their information all processors will know the entire input configuration and hence they can all compute the value of f on the given input. More formally, the algorithm is as follows. For any sequences of bits I, J let IJ denote the concatenation of I and J. Let I_p^i denote the input to processor p at the ith step of the computation. Initially I_p^0 is the input bit to processor p.

Algorithm for processor p:
initialize: I_p^0 is the input bit to processor p;
for $i := 0,\ldots,n-1$ **do**
  send message I_p^i to p's neighbor q along the ith link
  let I_q^i be the message received by p from p's neighbor q along the ith link and
  put $I_p^{i+1} := I_p^i I_q^i$;
od;
output $f(I_p^n)$

Let $I_p = I_p^n$ be the sequence obtained by processor p at the nth stage of the above algorithm. Let p, q be any two processors of the hypercube. Clearly, there is a unique flipping automorphism ϕ satisfying $\phi(p) = q$, namely $\phi = \phi_S$, where $i \in S$ if and only if $p_i \neq q_i$. In view of the above algorithm it is clear that this automorphism will map the input configuration I_p to the input configuration I_q. Hence $f(I_p) = f(I_q)$. This proves the correctness of the algorithm.

To study its complexity, let $T(N)$ be the number of bits transmitted in order that at the end of the computation all the processors in the hypercube know the input of the entire hypercube. By performing a computation on each of the two $n-1$-dimensional hypercubes we obtain that their nodes will know the entire input corresponding to their nodes in $T(N/2)$ bits. The total number of bits transmitted in this case is $2 \cdot T(N/2)$. The final exchange transmission consists of $N/2$ bits being transmitted by $N/2$ nodes to their $N/2$ corresponding other nodes, for a total of $2 \cdot N/2 \cdot N/2 = N^2/2$. Hence we have proved that $T(N) \leq 2 \cdot T(N/2) + N^2/2$. It follows that $T(N) \leq N^2$, as desired. □

Contrasting oriented and unlabeled hypercubes we have the following result.

Theorem 4.4 *For $n \geq 2$, there exist boolean functions $f \in B_N$, $N = 2^n$, computable on the oriented hypercube but not computable on the unlabeled hypercube Q_n.*

Proof. Define the boolean function f on inputs $< b_x : x \in Q_n >$ as follows. The value of f is 0 if for all adjacent nodes x, y with edge labeled by 1, $b_x = b_y$, otherwise it is equal to 1. More formally,

$$f(< b_x : x \in V >) = \begin{cases} 0 & \text{if } \forall x, y(\mathcal{L}(x,y) = 1 \Rightarrow b_x = b_y) \\ 1 & \text{otherwise.} \end{cases}$$

It is easy to see that f is kept invariant by all flipping automorphisms of Q_n but this is not true for any permuting automorphism ϕ_σ such that $\sigma(1) > 1$ (such an automorphism will also move the label). It follows that $F_n \subseteq S(f)$, but $F_n \cdot P_n \not\subseteq S(f)$, where $S(f)$ is the group of permutations in S_N that keep the boolean function f invariant under all inputs [CK89]. □

As a matter of fact we can prove a better result with a more difficult proof. There is a boolean function $f \in B_N$ such that $S(f) = F_n$. For this we need the representation theorem for permutation groups given in [CK89]. The exact value of the cycle index of both groups F_n and $F_n \cdot P_n$ is computed in [Har63].

Regarding symmetric functions on the oriented hypercube we have the following result.

Theorem 4.5 *On the oriented hypercube Q_n, every symmetric function can be computed in $O(N \cdot \log^2 N)$ bits. Moreover the threshold function Th_k can be computed in $O(N \cdot \log N \cdot \log k)$ bits, where $k \leq N$.*

Proof. The idea of the proof of theorem 4.3 can be used to compute the threshold function Th_k. We employ exactly the same algorithm, however in this case, the processors need only transmit the minimum between k and the number of 1s they have encountered so far, which requires at most $\log k$ bits. Consequently we obtain the inequality $T(N) \leq 2 \cdot T(N/2) + N \cdot \log k$. It follows that $T(N) \leq N \cdot \log N \cdot \log k$, as desired. Symmetric functions are handled in the same way. In each stage the processors transmit the exact number of 1s encountered. □

Clearly, OR_N can be computed in $O(N \cdot \log N)$ bits. The same bit complexity holds for the parity function. Observe that it is sufficient for the processors to transmit the bit 0 if the number of 1s seen so far is even, and 1 otherwise. Note that computing any symmetric function requires $\Omega(N \cdot \log N)$ bits. The lower bound proof given in [ASW85] works here as well. It merely uses the fact that the diameter of the network is $\log N$. Thus the algorithm of theorem 4.5 is optimal to within a factor of $O(\log N)$ for arbitrary symmetric functions and is exactly optimal for the functions OR_N and parity.

5 Conclusion 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 2 provides such an algorithm for any unlabeled network $\mathcal{N}$ with bit complexity $O(N^4 \cdot \delta \cdot d^2 \cdot \log N)$. It would be interesting however if we could improve on this bit complexity.

Surprisingly enough we have been able to find very efficient algorithms for computing symmetric functions on the class of distance regular networks (theorem 3.1). Nevertheless these algorithms do not seem to generalize to arbitrary unlabeled networks. Recently, using a different approach, algorithms for the case of symmetric functions have been found which are more efficient for general unlabeled networks than those suggested by theorem 2.2 (see [KKvdB89]).

An interesting special case is that of the hypercube network. Based upon the results of [ASW85] for unlabeled and oriented rings and [BB89] for oriented tori we conjecture that there are more efficient algorithms for computing boolean functions on the unlabeled and oriented hypercube than those suggested by theorem 4.1 and theorem 4.3 respectively.

There have been few studies in the literature regarding lower bounds. The only network for which this question has been studied extensively is the ring [MW86], [AAHK88], [DG87]. [PKR84] studies the question for the extrema finding function but relies on specific properties of this function. [YK87a] give lower bounds for the message complexity of computing boolean functions for broad classes of networks. However, very little is known about the bit complexity of boolean functions on the anonymous torus or hypercube, not to mention the general case of unlabeled networks.

Another interesting question concerns the group theoretic characterization of the class $\mathcal{F}^{\mathcal{N}[\mathcal{L}]}$ of functions which are computable on the network $\mathcal{N}[\mathcal{L}]$. As discussed in the introduction such characterizations are possible for the case of rings (oriented or not), oriented tori, as well as oriented hypercubes. However nothing seems to be known for the case of unlabeled tori or hypercubes. We know that for all networks $\mathcal{N}[\mathcal{L}]$, $\mathcal{F}^{\mathcal{N}[\mathcal{L}]} \subseteq \{f \in B_N : Aut(\mathcal{N}[\mathcal{L}]) \leq S(f)\}$. However for which networks is it true that $\mathcal{F}^{\mathcal{N}} = \{f \in B_N : Aut(\mathcal{N}) \leq S(f)\}$? For example can we show whether or not $\mathcal{F}^{Q_n} = \{f \in B_N : F_n \cdot P_n \leq S(f)\}$? It is interesting to note that [YK87a] gives an example of a network and a labeling such that this equality does not hold.

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 [AS88]. For general networks [SS89] 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.

6 Acknowledgements

We are grateful to L. Meertens for many fruitful conversations. C. Attiya and T. Tsantilas were very helpful with the bibliography.

References

[AAHK88] Karl Abrahamson, Andrew Adler, Lisa Higham, and David Kirkpatrick. Randomized evaluation on a ring. In Jan van Leeuwen, editor, *Distributed Algorithms, 2nd International Workshop, Amsterdam, The Netherlands, July 1987*, pages 324 – 331, Springer Verlag Lecture Notes in Computer Science, Heidelberg, 1988.

[Ang80] Dana Angluin. Local and global properties in networks of processors. In *12th Annual ACM Symposium on Theory of Computing*, pages 82 – 93, 1980.

[AS88] Hagit Attiya and Mark Snir. *Better Computing on the Anonymous Ring.* Technical Report RC 13657 (number 61107), IBM T. J. Watson Research Center, November 1988. 33 pages.

[ASW85] Chagit Attiya, Mark Snir, and Manfred Warmuth. Computing on an anonymous ring. In *4th Annual ACM Symposium on Principles of Distributed Computation*, pages 196 – 203, 1985.

[BB89] Paul W. Beame and Hans L. Bodlaender. Distributed computing on transitive networks: the torus. In B. Monien and R. Cori, editors, *6th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, pages 294–303, Springer Verlag Lecture Notes in Computer Science, Heidelberg, 1989.

[Big74] Norman Biggs. *Algebraic Graph Theory.* Cambridge University Press, 1974.

[Cam83] Peter J. Cameron. Automorphism groups of graphs. In Lowell W. Beineke and Robin J Wilson, editors, *Selected Topics in Graph Theory, Volume 2*, chapter 4, pages 89 – 127, Academic Press Inc., 1983.

[CK89] Peter Clote and Evangelos Kranakis. Boolean functions invariance groups and parallel complexity. In *4th Annual IEEE Symposium on Structure in Complexity Theory*, 1989.

[DG87] P. Duris and Z. Galil. Two lower bounds in asynchronous distributed computation. In *Proceedings 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 326 – 330, 1987.

[Har63] Michael A. Harrison. The number of transitivity sets of boolean functions. *J. Soc. Indust. Appl. Math.*, 11(3):806 – 828, September 1963.

[KKvdB89] E. Kranakis, D. Krizanc, and J. van der Berg. Computing symmetric functions on anonymous networks. 1989. unpublished manuscript.

[MW86] S. Moran and M. Warmuth. Gap theorems for distributed computation. In *5th Annual ACM Symposium on Principles of Distributed Computation*, pages 131 – 140, 1986.

[PKR84] J. Pachl, E. Korach, and D. Rotem. A new technique for proving lower bounds for distributed maximum finding algorithms. *J. of the ACM*, 31(4):905 – 918, October 1984.

[SS89] B. Schieber and M. Snir. Calling names on nameless networks. In *8th Annual ACM Symposium on Principles of Distributed Computation*, pages 319–328, 1989.

[YK87a] M. Yamashita and T. Kameda. *Computing on an Anonymous Network.* Technical Report 87-15, Laboratory for Computer and Communication Research, Simon Fraser University, 1987. 66 pages.

[YK87b] M. Yamashita and T. Kameda. *Computing on an Anonymous Network.* Technical Report 87-16, Laboratory for Computer and Communication Research, Simon Fraser University, 1987. 27 pages.