Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

E. Kranakis, D. Krizanc

Computing boolean functions on anonymous hypercube networks (Extended abstract)

Computer Science/Department of Algorithmics & Architecture Report CS-R9040 August

Computing Boolean Functions on Anonymous Hypercube Networks (Extended Abstract)

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

Danny Krizanc(2) (krizanc@cs.rochester.edu)

(1) Centrum voor Wiskunde en Informatica (CWI)
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 (i.e. total number of bits transmitted) of computing boolean functions on anonymous oriented hypercubes. We characterize the class of boolean functions computable in the anonymous oriented hypercube as exactly those boolean functions which are invariant under all bit-complement automorphisms of the hypercube and provide an algorithm for computing all such functions with bit complexity $O(N \cdot \log^4 N)$. Thus among all studied oriented networks (rings, tori, etc) the hypercube seems to achieve "optimal" bit complexity for a given number of nodes.

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

Report CS-R9040
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1 Introduction

The problem of characterizing the computable boolean functions in a given network of anonymous processors and of providing efficient algorithms for computing all such functions in the network has been considered in the literature for the case of rings [ASW85] and tori [BB89]. However, despite its versatility very little seems to be known concerning the above problem on the hypercube. The present paper is devoted to the construction of a new efficient algorithm for the computation of boolean functions on an oriented anonymous network consisting of identical processing elements which are connected in a hypercube topology. We show how to achieve bit complexity $O(N \cdot \log^4 N)$ for the total number of bits transmitted in computing boolean functions on such a network.

1.1 Assumptions

We make the following assumptions regarding the network and its 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 (i.e., they do not know either the identities of themselves or of the other processors),
3. the processors are identical (i.e., they all run the same algorithm),
4. the processors are deterministic,
5. the network is asynchronous,
6. the network is oriented (by orientation we mean a global, consistent labeling of the network links),
7. the network links are FIFO.

1.2 Labeled and Oriented Hypercubes

Let $Q_n = (V, E)$ be the n-dimensional hypercube. The node set V consists of all bit sequences $(x_1, \ldots, x_n)$ of length n and the edge set E consists of all pairs of nodes differing in exactly one component. By a labeling of Q_n we understand a function that for all nodes $v \in V$, associates the values $1, 2, \ldots, n$ 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 Q_n has an associated labeling $\mathcal{L}$ then it is called a labeled hypercube and is denoted by $Q_n[\mathcal{L}]$. Otherwise it is called an unlabeled hypercube. If we want to emphasize that a certain labeling is known to all processors of the network then we call the labeling an orientation.

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. $\mathcal{L}(x, y) = \mathcal{L}(y, x) = i$. In the sequel we will refer to a hypercube with this labeling as an oriented hypercube.

An *automorphism* of a network is a permutation of the node set of the network which preserves both incidence and labeling. 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 $f \in B_N$ (the set of Boolean functions on N variables) is invariant under a permutation $\sigma \in S_N$ if for all inputs $x_1, \ldots, x_N$,

$$f(x_1, \ldots, x_N) = f(x_{\sigma(1)}, \ldots, x_{\sigma(N)}) \in \{0, 1\}.$$

We denote by $S(f)$ the set of permutations in S_N that leave f invariant. The automorphism group of a network provides a necessary (but not always sufficient [YK88]) 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. 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 $Aut(\mathcal{N}[\mathcal{L}])$.

Of particular interest in the case of the oriented hypercube are the *bit-complement* automorphisms that complement 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). Let F_n denote the group of bit-complement automorphisms of Q_n. It is easy to show the following:

Theorem 1.1 *The group of automorphisms of the oriented hypercube Q_n is exactly the group F_n of bit-complement automorphisms.* ▮

1.3 Related Literature

For any boolean function f on N variables let $S(f)$ be the group of permutations that leave f invariant on all inputs. For the case of the ring [ASW85] gives algorithms both for the oriented and unoriented ring with bit complexity $O(N^2)$. In the oriented case they show that a boolean function $f \in B_N$ is computable if and only if $S(f) \geq C_N$, while in the unoriented case if and only if $S(f) \geq D_N$ (where C_N, D_N are the cyclic and dihedral groups on N letters, respectively). For the oriented torus [BB89] give an algorithm with bit complexity $O(N^{1.5})$ and show that a boolean function $f \in B_N$ is computable if and only if $S(f) \geq C_{\sqrt{N}} \otimes C_{\sqrt{N}}$. A characterization of the computable functions in a general anonymous network was given by [YK88] but their algorithm was of exponential bit complexity. The first algorithm for general networks with polynomial bit complexity was given by [KKvdB90]. Despite the strong interest in hypercubes very little is known except for the case of computing symmetric functions. For this case, [KKvdB90] gives two "general purpose" algorithms, one based on Markov chains and having bit complexity $O(N \cdot \log^4 N)$ and another taking advantage of the distance regular topology of the hypercube and having bit complexity $O(N \cdot \log^3 N)$.

1.4 Outline and Results of the Paper

In this paper we develop a new efficient algorithm for computing all boolean functions (which are computable in the hypercube) with bit complexity $O(N \cdot \log^4 N)$. In the first part of the paper we concentrate on the problem of characterizing the class of boolean functions which are computable on the oriented hypercube. This leads us to a simple algorithm with bit complexity $O(N^2)$. In the second part of the paper we use ideas of the first part and elementary results from group theory in order to develop a more sophisticated algorithm with bit complexity $O(N \cdot \log^4 N)$. In addition we show that symmetric functions on a oriented hypercube are computable in $O(N \cdot \log^2 N)$ bits Here is a table comparing existing results on the complexity of computing boolean functions on various anonymous networks.

Network	Bit Complexity	Paper
(Un)Oriented Rings	$O(N^2)$	[ASW85]
Oriented n-tori, n constant	$O(N^{1+1/n})$	[BB89]
Oriented hypercubes	$O(N \cdot \log^4 N)$	This paper

2 Initial Algorithm

In this section we characterize the class of boolean functions which are computable in the oriented hypercube in terms of its group of automorphisms and provide an algorithm with bit complexity $O(N^2)$ for computing all such functions. We can prove the following theorem for the previously defined natural orientation of the hypercube.

Theorem 2.1 *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 bit-complement 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 bit-complement 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 $i$th link
    **let** $I_q^i$ be the message received by $p$ from $p$'s neighbor $q$ along the $i$th link and
```

```
        put $I_p^{i+1} := I_p^i I_q^i$;
    od;
    output $f(I_p^n)$
```

To prove the correctness of the algorithm it must be shown that all processors output the same correct bit, i.e. for all processors p, q, $f(I_p^n) = f(I_q^n)$. 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 bit-complement automorphism ϕ satisfying $\phi(p) = q$, namely $\phi = \phi_S$, where $i \in S$ if and only if $p_i \neq q_i$. Now it can be shown that this automorphism will map processor p's view of the input, I_p, to the view of processor q, I_q. For any sequence $b_x b_{x'} \cdots$ of bits indexed by elements $x, x', \ldots \in Q_n$ define

$$\phi(b_x b_{x'} \cdots) = b_{\phi(x)} b_{\phi(x')} \cdots .$$

We can prove by induction on $i \leq n = \log N$ that $\phi(I_p^i) = I_{\phi(p)}^i$. This is clear for $i = 0$. Assume the result true for i. Let p', q' be p's and q's neighbors along the ith edge, respectively. Then by definition we have

$$I_p^{i+1} = I_p^i I_{p'}^i \text{ and } I_q^{i+1} = I_q^i I_{q'}^i.$$

Since ϕ is a bit-complement automorphism and p, p' are connected via the ith edge it follows that $\phi(p) = q$ and $\phi(p') = q'$. Using the induction hypothesis $\phi(I_p^i) = I_{\phi(p)}^i$ we obtain

$$\phi(I_p^{i+1}) = \phi(I_p^i)\phi(I_{p'}^i) = I_q^i I_{\phi(p')}^i = I_q^{i+1} = I_{\phi(p)}^{i+1}$$

This completes the inductive proof. It follows now that $\phi(I_p) = I_q$ which implies that $f(I_p) = f(I_q)$, since f is invariant under the bit-complement automorphisms of Q_n.

To study the bit complexity of the above algorithm, 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 2.2 *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 bit-complement automorphisms of Q_n but this is not true for any bit-permuting automorphism ϕ_σ such that $\sigma(1) \neq 1$ where

$$\phi_\sigma(x_1, \ldots, x_n) = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$$

since 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. ∎

3 Main Algorithm

In this section we make several alterations to the previous algorithm and show how to improve the complexity bound to $O(N \cdot \log^4 N)$, for each boolean function $f \in B_N$ which is computable in the hypercube. In all our subsequent discussions we use the notation and terminology given in the previous section. As before the new algorithm is also executed in $n = \log N$ steps, one step per dimension. However, now we take advantage of the fact that the transmitted views I_p^i provide information to p about the rest of "its hypercube". The main ingredients of the new algorithm are the following.

- We introduce a leader election mechanism which for each $i \leq \log N$ elects leaders among the processors with lexicographically maximal view at the ith step of the algorithm.
- We use elementary results from the theory of finite permutation groups [Wie64] in order to introduce a coding mechanism of the views; leaders at the $(i-1)$st step exchange the encoded versions of their views I_p^{i-1}; upon receipt of the encoded view they recover the original view sent and elect new leaders for the ith step.
- The leader election and coding mechanisms help keep low the number of bits transmitted during the ith step of the algorithm to $O(N \cdot i^3)$ bits.

The technical details of the above description will appear in the sequel. We begin with some preliminary lemmas that will be essential in the proof of the main theorem.

Lemma 3.1 *If $I_p = I_q$ then the hypercube as viewed from p is identical to the hypercube as viewed from q. More formally, for each p let $I_p =< b_x : x \in N >$. If $I_p = I_q$ and $\phi = \phi_S$, where $S = \{i \leq n : p_i \neq q_i\}$, then $\forall x \in Q_n(b_x = b_{\phi(x)})$.*

Proof. Indeed, notice that since $q = \phi(p)$

$$I_p = I_q = I_{\phi(p)} = \phi(I_p),$$

which proves the result. ∎

Lemma 3.2 *Let I be a fixed sequence of bits of length 2^n. Then the number of processors p such that $I_p = I$ is either 0 or a power of 2. Moreover the set of processors p such that $I_p = I$ can be identified with a natural group of bit-complement automorphisms.*

Proof. Let $\mathcal{J}$ be the set of processors q satisfying $I_q = I$ and assume that $\mathcal{J} \neq \emptyset$. Let p be an arbitrary but fixed element of $\mathcal{J}$. It is clear that for each $q \in \mathcal{J}$ there is a unique bit-complement automorphism $\phi_q \in F_n$ such that $\phi_q(p) = q$. It follows that the set

$$\mathcal{G} = \{\phi_q : q \in \mathcal{J}\}$$

is equipotent to $\mathcal{J}$. It can be shown easily that

$$\mathcal{G} = \{\phi \in F_n : \phi(p) \in \mathcal{J}\}.$$

We now show that $\mathcal{G}$ is a subgroup of F_n. Assuming this it would follow from the well-known theorem of Lagrange that the order of this group must divide the order of F_n which is exactly 2^n. Hence $|\mathcal{G}|$ is a power of 2. It remains to show that $\mathcal{G}$ is a subgroup of F_n. We show that it is closed under composition. Let $\phi, \psi \in \mathcal{G}$, i.e. by definition we have that

$$I_{\phi(p)} = I_{\psi(p)} = I.$$

Then we have that

$$I_{\phi(\psi(p))} = \phi(I_{\psi(p)}) = \phi(I_p) = I_{\phi(p)} = I.$$

Since the identity element is in $\mathcal{G}$, it follows that the latter is a group. It also remains to show that $\mathcal{G}$ is independent of the choice of the element $p \in \mathcal{J}$. Indeed, let $\mathcal{G}'$ be defined like $\mathcal{G}$ but using another element $p' \in \mathcal{J}$. Let ψ be a bit-complement automorphism such that $\psi(p) = p'$. Elementary calculations and the fact that $\mathcal{G}$ is an abelian group show that

$$\mathcal{G}' = \psi^{-1}\mathcal{G}\psi = \mathcal{G}.$$

This proves the first part of the lemma.

To prove the second assertion we note that F_n can be identified with an n-dimensional vector space over the finite field $Z_2 = \{0, 1\}$ of two elements. The standard basis of this vector space consists of the bit-complement automorphisms

$$\phi_{\{1\}}, \phi_{\{2\}}, \ldots, \phi_{\{n\}}.$$

Any other bit-complement automorphism ϕ_S can be written as the sum (which in this case is the regular composition of functions) of the automorphisms $\phi_{\{i\}}$, where $i \in S$. As a vector subspace $\mathcal{G}$ has a base consisting of a fixed number of bit-complement automorphisms. This proves the lemma. ∎

Clearly the group $\mathcal{G}$ defined in lemma 3.2 depends on the string I. However we avoid mentioning it explicitely in $\mathcal{G}$ in order to avoid unnecessary notational complications.

Lemma 3.3 *If $|\mathcal{G}| = 2^l$ then I can be coded with a string of length 2^{n-l} and l bit-complement automorphisms.*

Proof (Outline). We continue using the notation of lemma 3.2. The group $\mathcal{G}$ defined above has a natural action on the hypercube Q_n. For each $x \in Q_n$ let $x^{\mathcal{G}}$ be the orbit of x under $\mathcal{G}$, i.e.

$$x^{\mathcal{G}} = \{\phi(x) : \phi \in \mathcal{G}\}.$$

For each x the stabilizer $\mathcal{G}_x$ of $\mathcal{G}$ under x is the identity group, where the stabilizer group [Wie64] is defined by

$$\mathcal{G}_x = \{\phi \in \mathcal{G} : \phi(x) = x\}.$$

By the well-known stabilizer theorem [Wie64]

$$|\mathcal{G}_x| \cdot |x^{\mathcal{G}}| = |\mathcal{G}|.$$

Since $|\mathcal{G}_x| = 1$ we obtain that all the orbits of $\mathcal{G}$ have exactly the same size, namely $|\mathcal{G}| = 2^l$, and since $|Q_n| = 2^n$, there are exactly

$$\frac{2^n}{|\mathcal{G}|} = 2^{n-l}$$

pairwise disjoint orbits.

The above discussion gives rise to the following "coding" algorithm which can be applied by the processors concerned in order to code the given configuration I with a new (generally shorter) string. Each processor that knows I can execute the following "coding algorithm" (i.e. processor p applies this algorithm to the string $I = I_p^n$).

Coding Algorithm:

Input: $I = < b_x : x \in Q_n >$ is the given configuration, where b_x is the bit corresponding to processor x.

1. Compute the group $\mathcal{G}$ of bit-complement automorphisms ϕ such that

$$\forall p \in Q_n(I_p = I \Rightarrow I_{\phi(p)} = I).$$

   Assume that l is such that $|\mathcal{G}| = 2^l$.

2. Compute a set of l generators, i.e. bit-complement automorphisms $\phi_1, \ldots, \phi_l$ which generate the group $\mathcal{G}$.

3. Compute the set of orbits of $\mathcal{G}$ in its natural action on Q_n. There are 2^{n-l} such orbits. For each orbit the processors choose a representative of the orbit in some canonical way, say lexicographically minimal; let $x(1), x(2), \ldots, x(2^{n-l})$ be the representatives chosen. Next the processor arranges them in increasing order according to the lexicographic order $\prec$, i.e. $x(1) \prec x(2) \prec \ldots \prec x(2^{n-l})$.

4. The code of I is defined to be the sequence $< I'; \phi_1, \phi_2, \ldots, \phi_l >$, where I' is the sequence of bits of length 2^l given by

$$I' := b_{x(1)} b_{x(2)} \cdots b_{x(2^{n-l})}$$

   and

$$\phi_1, \phi_2, \ldots, \phi_l$$

   is a sequence of bit-complement automorphisms generating the group $\mathcal{G}$.

Output: $< I'; \phi_1, \phi_2, \ldots, \phi_l >$.

It remains to prove that a processor can reconstruct I from its encoding. To do this it executes the following decoding algorithm.

Decoding Algorithm:

Input: $< I'; \phi_1, \phi_2, \ldots, \phi_l >$, where I' is a string of length 2^{n-l} and $\phi_1, \phi_2, \ldots, \phi_l$ are bit-complement automorphisms.

1. Let $\mathcal{G}$ be the group generated by these automorphisms. Compute the set of orbits of $\mathcal{G}$ in its natural action on Q_n. There are 2^{n-l} such orbits. For each orbit choose as representative of the orbit the lexicographically minimal string in the orbit. Let $x(1), x(2), \ldots, x(2^{n-l})$ be the representatives chosen. Next the processor arranges them in increasing order according to the lexicographic order $\prec$, i.e. $x(1) \prec x(2) \prec \ldots \prec x(2^{n-l})$.

2. The previous coding algorithm guarantees that $I' = b_{x(1)}b_{x(2)} \cdots b_{x(2^{n-i})}$. Hence we can "fill-in" the remaining bits to form the string I since $b_x = b_y$ for x, y in the same orbit.

Output: I.

Indeed, by definition of the group $\mathcal{G}$ we have that for all $\phi \in \mathcal{G}$, $\phi(I) = I$. Hence by lemma 3.1

$$\forall x \in Q_n \forall \phi \in \mathcal{G}(b_x = b_{\phi(x)}),$$

where $I = < b_x : x \in Q_n >$. This explains why the decoding algorithm works. The rest of the details are left to the reader. ∎

Now we can prove the following theorem which significantly improves the upper bound of theorem 2.1.

Theorem 3.1 *There is an algorithm for computing every boolean function $f \in B_N$ (which is invariant under all bit-complement automorphisms) on the oriented hypercube Q_n, $N = 2^n$, with bit complexity $O(N \cdot \log^4 N)$.*

Proof (Outline). For each fixed string $x = x_{i+1} \cdots x_n$ of bits of length $n - i$ let

$$Q_i(x) = \{u_1 \cdots u_i x : u_1, \ldots, u_i \in \{0, 1\}\}.$$

For each processor p represented by the sequence $p_1 \cdots p_n$ of bits the ith hypercube of p is defined to be $Q_i(p_{i+1} \cdots p_n)$. Clearly we have that

$$Q_i(x) = Q_{i-1}(0x) \cup Q_{i-1}(1x).$$

Initially, I_p^0 = "input bit to processor p" and each processor declares itself leader of its 0-dimension hypercube $Q_0(p) = \{p\}$. The leaders at the ith step of the algorithm are among those processors whose "view" I_p^i of their ith hypercube is lexicographically maximal among the set of strings I_q^i with q an active leader (defined below). Assume by induction that we have elected leaders for the $(i-1)$th stage of the algorithm and that each processor has a path to such a leader along its hypercube with edges $\leq i - 1$. We show how to extend these assumptions to the ith stage of the algorithm. Thus the ith stage of the new algorithm consists of the following steps.

1. The leader-processors send their encoded views of their hypercube to their neighbors along the ith dimension.

2. The processors of the opposite hypercube receiving the views route them to their leaders. (All the processors know routes to their leaders along their hypercube; hence they can transmit the view received along such a route, say the lexicographically minimal one.) Leaders that receive such encoded views become "active leaders"; they decode the messages as in lemma 3.3 compute the corresponding views of their neighbors along their ith edge and append it to their own view thus forming views at step i. To compute the view of their neighbors along their ith edge each leader ℓ executes the following algorithm

   (a) Let ℓ's neighbor along the ith edge be p and let $1 \leq k_1, \ldots, k_r \leq i - 1$ be a path along p's subcube leading to a leader ℓ' in this subcube (by the leader position algorithm and the induction hypothesis we can assume that such a path is known to p). By the previous argument the view $I_{\ell'}^{i-1}$ of ℓ' is known to ℓ. Now ℓ requests this path from its neighbor p.

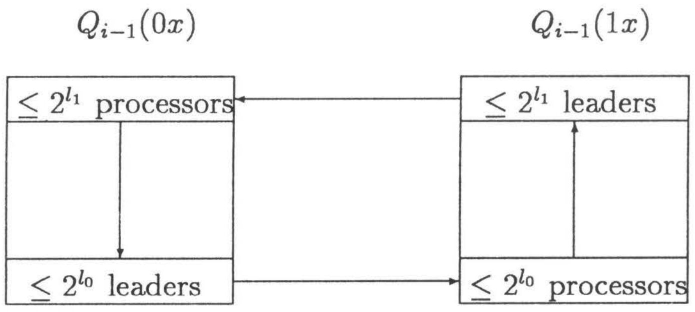

Figure 1: Exchange of Views among Leaders in Hypercube $Q_i(x)$

   (b) Since $\phi_{\{k_1,\ldots,k_r\}}(\ell') = p$ it is clear that ℓ can compute p's view via the identity

$$\phi_{\{k_1,\ldots,k_r\}}(I_{\ell'}^{i-1}) = I_p^{i-1}.$$

3. Active leaders send an "Active Leader" message to all their edges labeled $\leq i-1$ in order to inform the rest of the processors they are active (this is similar to the leader position algorithm below). New leaders can now be elected among the active leaders which have lexicographically maximal view. These leaders use the coding algorithm previously described in order to code their views. They also inform the rest of the processors of the hypercube $Q_i(x)$ of their position in the hypercube (see leader-position algorithm below).

4. Return to 1 and iterate, for $i = 1, 2, \ldots, \log N$.

The above algorithm is depicted in figure 1.

To inform the rest of the processors of their position the leaders execute the following algorithm.

Leader-position Algorithm

```
1. Initialize $L := LEADER$.
2. For $j = 0$ to $i$ do
    (a) send $L$ to all neighbors in subcube
    (b) processor receiving $L$ via edge labeled $k$ appends $k$ to $L$, i.e. $L :=< L, k >$.
```

After i steps the processors will receive paths, like

$$< LEADER, k_1, k_2, \ldots, k_i >$$

indicating that a leader can be found along the path indicated by the sequence

$$k_i, k_{i-1}, \ldots, k_1$$

The total bit complexity of this algorithm is $O(2^i \cdot i^3)$.

Now we estimate the bit complexity of the algorithm. The coding and decoding algorithms are "internal" and do not contribute anything to the total bit complexity. Suppose there are $\leq 2^l$ leaders elected at the ith step of the algorithm, there exists a message w of length 2^{i-l} and a sequence of $l \leq i$ bit-complement automorphisms of the hypercube Q_i which "code" the view I_p^i. Since only the leaders transmit messages at the ith step while the rest of the processors are "routing" messages to the leaders (processors are always at a distance $\leq i$ from a leader, since the diameter of the ith hypercube is i), the total bit complexity at the ith step of the algorithm is $O(2^i \cdot i^3)$ (since each encoded view consists of at most i bit-complement automorphisms and each bit-complement automorphism can be coded with i bits). Clearly this algorithm is applied to 2^{n-i} subcubes simultaneously. Since the algorithm is iterated $\log N$ times it follows that the bit complexity of the new algorithm is

$$\sum_{i=1}^{\log N} 2^{n-i} \cdot O(2^i \cdot i^3) = O(N \cdot \log^4 N).$$

This proves the theorem. ∎

4 Symmetric Functions

For the case of symmetric functions the algorithm given in section 2.1 may be modified to achieve an even better bit complexity.

Theorem 4.1 *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 2.1 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, the above can be used to compute the OR of N variables in $O(N \cdot \log N)$ bits. The same bit complexity holds for the parity function by just remembering wether the number of 1s is even or odd. The lower bound proof given in [ASW85] may be modified to show that any symmetric function requires $\Omega(N \cdot \log N)$ bits to compute on the hypercube. Thus the algorithm of theorem 4.1 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 Further Work

In this paper we developed an efficient algorithm for computing all computable boolean functions on the anonymous, oriented hypercube with bit complexity $O(N \cdot \log^4 N)$. For the case of symmetric functions this may be improved to $O(N \cdot \log^2 N)$ bits. Little seems to be known for the unlabeled, anonymous hypercube, except for the results of [KKvdB90] which gives algorithms computing symmetric functions with bit complexity $O(N \cdot \log^3 N)$ and arbitrary functions with bit complexity $O(N^4 \cdot \log^4 N)$.

Our algorithm was based on the coding-decoding mechanism of the views. In turn, the efficiency of this mechanism was based on the fact that that every permutation group $\mathcal{G}$ has $O(\log |\mathcal{G}|)$ generators. Thus it is possible to extend our main theorem to abelian groups. Furthermore, in the oriented torus on $N = m^n$ nodes, composed of oriented rings of size m there is an algorithm for computing all computable boolean functions with bit complexity $O(N^{1+2/n} \cdot \log^4 N / \log^3 m)$. Applications of this technique to more general anonymous Caley graphs are investigated in [KK90].

6 Acknowledgements

We are grateful to L. Meertens, J. Tromp and P. Vitányi for many fruitful conversations

References

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

[BB89] P. W. Beame and H. 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, 1989.

[KK90] E. Kranakis and D. Krizanc. Computing boolean functions on Caley graphs, 1990. In preparation.

[KKvdB90] E. Kranakis, D. Krizanc, and J. van der Berg. Computing boolean functions on anonymous networks. In M. S. Paterson, editor, *Proceedings of ICALP 90*, volume 443. Springer Verlag Lecture Notes in Computer Science, 1990.

[Wie64] H. Wielandt. *Finite Permutation Groups*. Academic Press, 1964.

[YK88] M. Yamashita and T. Kameda. Computing on an anonymous network. In *7th Annual ACM Symposium on Principles of Distributed Computation*, pages 117 – 130, 1988.