Optimal Elections in Labeled Hypercubes*

Paola Flocchini† Bernard Mans‡

Abstract

We study the message complexity of the *Election* Problem in Hypercube networks, when the processors have a "Sense of Direction", i.e., the capability to distinguish between adjacent communication links according to some globally consistent scheme. We present two models of Sense of Direction, which differ regarding the way the labeling of the links in the network is done: either by matching based on dimensions, or by distance along a Hamiltonian cycle.

In the dimension model, we give an optimal linear algorithm which uses the natural dimensional labeling of the communication links. We prove that, in the distance-based case, the graph symmetry of the hypercube is broken and, thus, the leader Election does not require a global maximum-finding algorithm: $O(1)$ messages suffice to select the leader, whereas the $\Theta(N)$ messages are required only for the final broadcasting. Finally, we study the communication cost of changing one orientation labeling to the other and prove that $O(N)$ messages suffice.

1 Introduction

In distributed systems, one of the fundamental control problems is the **Leader Election**; that is, the problem of moving the system from an initial situation where the nodes are in the same computational state, to a final situation where exactly one node is in a distinguished computational state (called *leader*) and all the others are in the same state (called *defeated*). The election process may be independently started by any subset of the processors. It is assumed that every processor P_i has a distinct identity id_i chosen from some infinite totally ordered set ID, and is only aware of its own identity (in particular, it does not know the identities of its neighbors).

*This work was done while both authors were visiting the School of Computer Science, Carleton University, Ottawa, Canada

†School of Computer Science, Carleton University, Ottawa, K1S 5B6 Canada, E-mail: flocchin@scs.carleton.ca

‡Dept. of Computer Science, James Cook University of North Queensland, Townsville, Qld 4811, Australia, E-mail: bernard@cs.jcu.edu.au

The leader election is widely used solution for distributed algorithms requiring one process to act as a coordinator, initiator, sequencer, or otherwise to perform some special role. The election problem occurs, for instance, in token-passing when the token is lost or the owner has failed; in such a case, the remaining processors elect a leader to issue a new token. Several other problems encountered in distributed systems can be solved by election; for example: *crash recovery* (a new server should be found to continue the service when the previous server has crashed), *mutual exclusion* (where values for election can be defined as the last time the process entered the critical section), *group server* (where the choice of a server for an incoming request is made through an election between all the available servers managing a replicated resource), etc.

Depending on the assumptions and the knowledge of the distributed system, the communication complexity changes. For an arbitrary network of N asynchronous processors and e bidirectional communication links, the Election problem requires $\Omega(e + N \log N)$ messages (e.g., [17]), and such a bound is achievable [4]. If *a priori* knowledge of the structure of the system is available to the processors, it is possible to exploit topological properties to reduce the message complexity to the smallest of two factors: the number of links e and the $(N \log N)$ factor. In particular, $\Theta(N \log N)$ bounds have been proved for the complete network [9, 10] in such a case. Obviously, such an approach does not yield any improvement in a topology where the number of edges e is of the same order as $N \log N$. This is the case of the Hypercube, and an $O(N \log N)$ message complexity can be trivially achieved.

It has become quite clear that a very important role is played by those factors which can be termed as Structural Information; that is, a priori knowledge available to the entities about the structure of the system. Currently, the study of the interaction between structural information and communication complexity is mostly implicit in results on distributed algorithms. It is crucial to identify those factors which are significant for the computability and the communication complexity of problems. This is all the more relevant for an attractive and successful topology such as the Hypercube. A form of structural information which has been identified as being significant for the computability and the communication complexity of the election problem is *Sense of Direction* [3, 18]. Sense of Direction refers to the capability of a processor to distinguish between adjacent communication lines, according to some globally consistent scheme.

After a brief presentation of Hypercube in Section 2 and Sense of Direction in Section 3, we present two algorithms for the Election problem according to the dimensional and the distance-based models, (Section 4). In Section 5 we study the complexity relationship between two models of "Sense of Direction" for the Hypercube and present an $O(N)$ message complexity algorithm to change from one Sense of Direction to another. Section 6 concludes and contains comments on related studies on this problem.

2 Hypercube

An n-dimensional (or N-node) *hypercube* network Q_n is represented by an undirected graph consisting of $N = 2^n$ vertices which can be labeled from 0 to $2^n - 1$ in such a way that there is an edge between any two vertices if and only if the binary representation of their labels differs by exactly one bit. The n-dimensional hypercube has $n2^{n-1}$ edges, and every node has degree n. Each edge is labeled at each incident node by the dimension of the bit of the identity in which they differ. The hypercube has node and edge symmetries. For any pair of edges (u, v) and (u', v') in a N-node hypercube Q, there is an automorphism σ of Q such that $\sigma(u) = u'$ and $\sigma(v) = v'$. Such an automorphism can be found for any permutation π on $\{1, 2, \cdots, n\}$ such that $\pi(k') = k$ where k and k' are the respective dimensions of (u, v) and (u', v'), [11]. The dimensions of the edges can be rearranged in any order by varying π without altering the network.

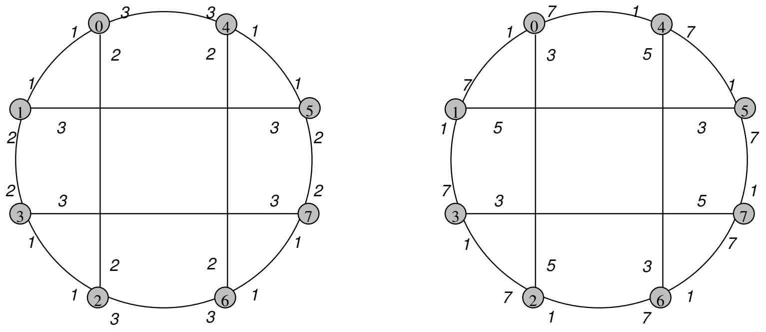
The computation model that we consider is a hypercube network of $N = 2^n$ asynchronous processors. Every processor P_i of the hypercube has a distinct value id_i chosen from some infinite totally ordered set ID. This value is different from the binary location identity introduced in the definition of the hypercube. A processor is aware only of its own value; in particular it does not know the identities of its neighbors. Every processor performs the same algorithm. We assume that the messages on each arc arrive with no error, within a finite but unpredictable time and in a FIFO order. The complexity measure studied is the maximum number of messages sent during any possible execution.

3 Sense of Direction

Sense of Direction has been identified as being significant for the computability and the communication complexity of many distributed problems [3, 18]. We have already observed that Sense of Direction refers to the capability of a processor to distinguish between adjacent communication lines, according to some globally consistent scheme.

In a ring network this property is usually referred to as orientation, which expresses the ability of the processor to distinguish between *left* and *right*, where *left* means the same to all processors. In a torus, a vertical orientation, with *up* and *down* labelings, is added. Sense of Direction in a complete network is the knowledge of a predefined Hamiltonian cycle and the existence of a label on each link at each processor, that represents the distance along this cycle to the processor at the other end of the link. A similar definition exists for chordal rings or circulant graphs (processors arranged on a ring on which extra chords with given distances are added [1]). This distance-based Sense of Direction can be generalized for arbitrary graphs by fixing a cyclic order of all the processors and labeling each incident link according to the distance in the cycle to the other node reached by this link; we denote this as **distance Sense of Direction** [1, 7, 12, 13]. The most natural labeling for a hypercube is the **dimensional Sense of Direction**. In this case each edge is labeled at each incident node by the dimension of the bit in which their location identities differ (figure 1 (a)).

The availability of Sense of Direction has been shown to have some positive impact on the message complexity of the Election problem. For instance, in arbitrary networks, the Election problem can be solved using $\Theta(N \log N)$ messages instead of $\Theta(e + N \log N)$, when a distance Sense of Direction is available [13]. In the complete graph, the availability of a distance Sense of Direction makes it possible to reduce the complexity from $\Theta(N \log N)$ to $\Theta(N)$ messages [12]; such an improvement can be achieved even if each node has Sense of Direction on just $O(\log N)$ of its incident arcs [1]. Similar results hold for circulant graphs or chordal rings with $O(\log \log N)$ incident arcs [8]. In the torus, an $O(N)$ messages algorithm has been given [15] when a dimensional Sense of Direction is available, it is not clear whether the problem can be solved with linear complexity when Sense of Direction is not available.



(a) (b)

Figure 1: Model of Sense of Direction for Hypercube: (a) Dimension, (b) Distance

In this paper, we consider the Election problem in the hypercube using two models of Sense of Direction, the *dimensional* and the *distance* model. In this latter case, a Hamiltonian cycle is the natural way to fix an ordering of all the nodes. Several different Hamiltonian cycles can be built in a hypercube, but the one based on the *binary reflected* Gray code [5, 6] is particularly used in the following. A handy notation of a Hamiltonian cycle in a hypercube corresponds to the list of the labels of the links traversed during its construction - this notation is effective since it ignores the identity of the processors visited. Such an N-tuple is classically called a *coordinate sequence* for the cycle. An example for a reflected coordinate cycle with $n = 4$ with the dimensional model is $\langle 1213121412131214 \rangle$

In order to follow a Hamiltonian cycle, a simple message which contains the rank in the cycle of the last processor visited is passed and increased; for instance, the first processor has to send the message through the link labeled 1 and the $4th$ processor which receives the message knows that it has to send it through the link labeled 3. It is worth noting that a Hamiltonian cycle in a n-dimensional hypercube can be built by induction on an $(n-1)$-dimensional hypercube: the new coordinate sequence A_n is obtained from the previous one A_{n-1} and the concatenation of the new value of degree n, i.e., $A_n = \langle A_{n-1} n A_{n-1} n \rangle$.

4 Election Algorithms for the Hypercube

The problem of electing a leader is one of the most widely studied in the literature on distributed computing. Starting from an initial configuration where all the processors are in the same state, the *Election* (or *Leader Finding*) problem consists of obtaining a final configuration where exactly one processor is in a *leader* state and all the other processors are in a *defeated* state. The Election process may be independently started by any subset of the processors.

In this paper, it is shown that the hypercube network, which has $O(\log N)$ arcs but is not a chordal ring, supports an Election Algorithm with $O(N)$ messages when a distance-based or dimensional Sense of Direction is available. This result answers a question raised when Attiya *et al* [1] proved that $O(N)$ messages suffice for the chordal ring with chords with distance-labeled chords $(1, 2, 2^2, \cdots, 2^{\lceil \log N \rceil}, N-1)$, which closely resembles a hypercube. In the following we show a linear election algorithm when a dimensional Sense of Direction is available; in section 4.2 we show that with a distance Sense of Direction the problem becomes trivial, mostly thanks to a graph property of symmetry.

4.1 Election Algorithm with the Dimensional Model

In this Section we give a complete presentation of the asynchronous distributed algorithm for the Election in the hypercube with a $\Omega(N)$ message complexity with a Dimensional Sense of Direction. It is worth noting that the processors do not have (or do not know) the binary node labeling from 0 to $2^n - 1$ which is usually assumed for hypercube (otherwise this provides a trivial solution to the problem).

The algorithm proceeds in phases of a tournament. The idea is to halve the number of competing processors on each phase such that on the overall $\log N$ phases, or steps, only a linear number of messages has been sent. This algorithm is original since it does not build a spanning tree. Initially all nodes are *Duellist*(0). The algorithm terminates after $(\log N)$th steps with a single Duellist *Duellist*($\log N$) and all the other nodes are *Seconds*. In every phase of the algorithm, each successful Duellist goes for the next duel by challenging its respective Duellist. The algorithm is based on repeated sequences of a duel (namely a combination of two cubes into a larger one) and, hence, takes $\log N$ steps (one per dimension).

At each step k, each Duellist has to challenge (send an attack) and to be challenged (receive an attack) by its respective Duellist in the rank of the tournament (the Duellist node in its cube image according to the dimension k). The two opposite Duellists handshake: the Duellist with the larger identity value that receives an attack from a Duellist with the smaller identity value wins the duel of step k and becomes Duellist of level $k+1$; conversely, the Duellist with the smaller identity value that receives an attack from Duellist with a larger identity value loses and becomes a *Second* of level k and keeps the path to its winner. Neither acknowledgment nor surrender messages are required. The task of a Duellist is to fight a duel, whereas the task of a Second is to relay an incoming attack to its Duellist.

The fundamental property used in the algorithm is that, at any step i, a Second of level k (with $k < i$) knows the *coordinate sequence* traversed by its Duellist (of level $k+1$), i.e., the Duellist against which it lost. The location of a Duellist of level $k+1$ is unambiguously known by the Second, and a shortest path between them can be computed locally by "compressing" the coordinate sequence received during the fatal attack: every pair of coordinates is eliminated (e.g., a sequence $\langle 123142 \rangle$ is equivalent to the sequence $\langle 34 \rangle$). We denote with $Duellist(s)$ and $Second(s)$ entities which, respectively, won and lost the duel of step s. When an attack from $Duellist(s)$ reaches a $Second(k)$ (with $k < s$), the Second forwards it to the node that will be known as the $Duellist(k+1)$. If this node is no longer a Duellist (i.e., has been defeated and became a Second), it forwards it to its respective $Duellist(k+2)$. The process is repeated until the $Duellist(s)$ is found. To reduce the amount of communication, an attack from $Duellist(s)$ reaching a $Duellist(k)$ (with $k < s$) is delayed until the $Duellist(k)$ either reaches the required level s, or becomes a Second $Second(k)$ able to forward the message to its respective $Duellist(k+1)$.

Local information used:

(i) $State_p : \{Duellist, Second, Leader\}$ is the state of a node (initially *Duellist*).

(ii) For each node p, Id_p is its identity, and $level_p$ is its level,

(iii) For each Second p, $NextDuellist$ denotes the path (the coordinate sequence) from the Second p at level $level_p$ to the Duellist of level $level_p + 1$.

Messages used:

(i) ATTACK: This message represents the challenge for a duel and contains: $(Id, lev,$-$source, dest)$, where *Id* is the identity of the initiator of the attack, *lev* is the step of the attack, *source* is the path (the coordinate sequence) from the attacking Duellist to the present node receiving the attack. This list is stored to know the path followed. Finally, *dest* is the path from the present owner of the message to a node target; it will be used to forward an attack by the shortest path between a Second to its Duellist.

(ii) LEADER: Final broadcast by the Leader to terminate and inform of its Id.

(iii) WAKEUP: An arbitrary number of processors can spontaneously start the execution of the algorithm; this is modeled by the reception of a message.

Functions used on coordinate sequence *list*:

(i) first(list) returns the first element of the list. Namely, this function gets the rank of the rightmost non-zero bit of the log N bits list.

(ii) $list \oplus i$ (resp. $\ominus$) updates the given path by adding (resp. eliminating) a label i to the list and compressing it. The compression guarantees that the coordinate sequence never exceeds log N labels and, hence, is represented by log N bits: a bit of rank i set to 1 means that the label i is present in the coordinate sequence, 0 otherwise. Namely, the operations $\oplus$ and $\ominus$ are identical: when applied on bit i of value b_i, they both set its value to $(1 - b_i)$.

procedure *Election*(p)
begin
/* initially - processor can be asleep */
(0) Upon RECEIPT of *(WAKEUP)* or any other message on any arc
  if message $\neq$ WAKEUP **then** delay message
  $State_p$:= $Duellist$; $level_p$:= 0
  $Id := Id_p$; $lev := 0$; $source$:= [] ; $dest$:= []
  SEND *ATTACK(Id,lev,source,dest)* on arc labeled 1
end WAKEUP

• If $State_p = Duellist$:

(1) Upon RECEIPT of *ATTACK(Id,lev,source,dest)*
 with ($lev = level_p$) on arc labeled r
  if ($Id_p > Id$ and $lev = n$) **then**
   $State_p := Leader$
   SEND(LEADER,Id_p) on all arcs
  endif
  if ($Id_p > Id$ and $lev < n$) **then**
   $level_p := level_p + 1$
   $Id := Id_p$; $lev := lev + 1$; $source$:= [] ; $dest$:= []
   SEND(ATTACK(Id,lev,source,dest)) on arc labeled $r + 1$
   accept delayed message with $lev = level_p$ if arrived
  endif
  if ($Id_p < Id$) **then**
   $State_p := Second$
   $NextDuellist := source \oplus r$
   accept all delayed messages if any
  endif
end ATTACK

(2) Upon RECEIPT of *ATTACK(Id,lev,source,dest)*
 with ($lev > level_p$) on arc labeled r
   delay message
end ATTACK

• If $State_p = Second$:

(3) Upon RECEIPT of *ATTACK(Id,lev,source,dest*)
 with ($dest \neq$ []) on arc labeled r
  $l := first(dest)$; $dest := dest \ominus l$; $source := source \oplus r$
  SEND(ATTACK(Id,lev,source,dest)) on arc labeled l
end ATTACK

(4) Upon RECEIPT of *ATTACK(Id,lev,source,dest*)
 with ($dest =$ []) on arc labeled r
  $dest := NextDuellist$
  $l := first(dest)$; $dest := dest \ominus l$; $source := source \oplus r$
  SEND(ATTACK(Id,lev,source,dest) on arc labeled l
end ATTACK

(5) Upon RECEIPT of *LEADER(LeaderId)* on arc labeled r
 Forall $k < r$
  SEND(LEADER(LeaderId)) on arc labeled k
end LEADER
end *Election*(p)

Step analysis Let $d(k-1,k)$ be the maximum distance between Duellist$(k-1)$ and Duellist(k), clearly $d(k-1,k)=k$. Let $F(i)$ be the maximum number of links that have been traversed at step i to go from Duellist(0) to Duellist$(i-1)$.

Obviously, in Step 1 the N Duellists(0) attack their corresponding Duellists(0), and, thus, each has to traverse only one link: $F(1)=1$. In Step 2, the $\frac{N}{2}$ Duellists(1) send an ATTACK; the attack can reach a Second(0) that has to forward it to Duellist(1), thus the maximum number of links to be traversed is $F(2)=1+d(0,1)=1+1=2$. In Step 3, the $\frac{N}{2^2}$ Duellists(2) send an ATTACK; the attack can reach a Second(0) that forwards it to the Duellist(2) through a Second(1), thus $F(3)=1+d(0,1)+d(1,2)=4$. For any step i, the $\frac{N}{2^{i-1}}$ Duellists$(i-1)$ initiates an ATTACK; the attack, in the worst case, has to follow the path: $\langle$Second(0) $\ldots$ Second$(i-2)$ Duellist$(i-1)$ $\rangle$, the number of links to be traversed is

$$F(i)=1+\sum_{k=1}^{i-1} d(k-1,k)=1+\sum_{k=1}^{i-1} k=1+\frac{i\cdot(i-1)}{2}$$

Full analysis

Theorem 4.1 *The total number of messages sent during the algorithm for the Election problem is at most*

$$7N-(\log N)^2-3\log N-7$$

Proof At any step i at most $\frac{N}{2^{i-1}}$ nodes send an ATTACK message. During each of the $\log N$ steps at most $F(i)$ forwarding messages are required to reach the target Duellist.

Thus, the total number M of messages is

$$\begin{aligned} M &= \sum_{i=1}^{logN} \frac{N}{2^{i-1}} \cdot F(i) = \sum_{i=1}^{logN} \frac{N}{2^{i-1}} \cdot \left(1+\frac{i\cdot(i-1)}{2}\right) \\ &= N\left(\sum_{i=1}^{logN}\frac{i^2}{2^i} - \sum_{i=1}^{logN}\frac{i}{2^i} + 2\sum_{i=1}^{logN}\frac{1}{2^i}\right) \end{aligned}$$

Since

$$\sum_{i=1}^{logN}\frac{i^2}{2^i} = \frac{6N-4\log N-(\log N)^2-6}{N} \tag{1}$$

$$\sum_{i=1}^{logN}\frac{i}{2^i} = \frac{2N-\log N-2}{N} \tag{2}$$

$$\sum_{i=1}^{logN}\frac{1}{2^i} = \frac{N-1}{N} \tag{3}$$

and, since exactly $(N-1)$ LEADER messages are required to broadcast the termination and leader identity, we prove the theorem. $\square$

Theorem 4.2 *Each message is composed of at most* $(\log Q + \log \log N + 2 \cdot \log N)$ *bits, where the identity of a node is at most* Q.

Proof Each ATTACK message contains the identity Id_p of the attacking Duellist, the level lev of the attack whose value is at most $\log N$, the location of the attacking Duellist ($\log N$ bits) and the location of the attacked Duellist ($\log N$ bits). A LEADER message contains only an identity. □

In the worst case, our tournament results in an $\Omega(\log^2 N)$ time per step (messages are passed serially), and a $\Theta(\log^3 N)$ overall time complexity.

Correctness Several properties are introduced for the correctness. In the following, numbers between parentheses refer to corresponding sections of the algorithm.

Lemma 4.1 *Let an ATTACK message contain a level* lev *initiated by a Duellist* p *of level* $level_p$, *then*
(i) the message has been originated through a link labeled lev *such that* $lev = level_p$,
(ii) after traversing the link through which it was originated, the message cannot traverse a link labeled d, *with* $lev < d$.

Proof (i) immediate from (1) ; (ii) by induction: since a Second never modifies the level of an ATTACK message (3) (4), and a Second never initiates a duel, the ATTACK traverses only links of smaller level. Therefore the ATTACK message cannot exit the subcube of degree lev to which it belongs. □

The next lemma guarantees that a Second knows the shortest path to reach its Duellist.

Lemma 4.2 *A Second(i) knows the shortest path (a coordinate sequence with minimum length) to reach its respective Duellist*$(i+1)$.

Proof The correctness of the path is guaranteed through the compression of the *source* list which stores the labels traversed links (3) and (4). By contradiction; assume that the length of the path is strictly greater than the shortest one, this would imply that at least two labels with the same direction exist in the coordinate path, which is forbidden by *compression*, $\oplus$ and $\ominus$. □

We now show that no infinite delays are introduced during the execution of the algorithm.

Lemma 4.3 *A deadlock cannot be introduced by the waiting that arises when some nodes must wait until some condition holds.*

Proof Since the message broadcast by the LEADER is forwarded immediately upon reception, only ATTACK messages may create a cycle. In Lemma 4.1 the partitioning of the subcube traversed by an ATTACK message has been shown and, thus, only cycles between two Duellists at the same level can be created. The only situation in which an entity is waiting for an event is when a Duellist a waits to be accepted by another Duellist b with $level_b < level_a$ to reach the respective level $level_a$, (2). The extreme case occurs when a chain of waiting processors such that $level_x < .. < level_b < level_a$ is created. By (0), the total ordering of the chain forbids the creation of a cycle in such a chain.

□

We now prove that an ATTACK message always reaches the target Duellist.

Lemma 4.4 *An ATTACK from a Duellist(i) ($i < \log N$) eventually reaches its respective Duellist(i).*

Proof By induction.

Initially each node is a Duellist(0) (Wake Up). The Lemma clearly holds for $i = 0$. In fact an ATTACK from a Duellist(0) reaches another Duellist(0) in exactly one message, through the link labeled "1".

Assume it holds for $j < i$; that is, suppose that any Duellist(j) with $j < i$ that sends an ATTACK eventually reaches another Duellist(j). We show that it also holds for i.

Consider a Duellist(i) that sends an ATTACK through the link labeled $i + 1$. Since this attack remains inside the $(i + 1)$-cube (Lemma 4.1 (ii)), it reaches either
(a) a Duellist(i') (with $i' \leq i$) or (b) a Second(i'), (with $i' < i$).

In case (a), if $i' = i$ the lemma is proved. If $i' < i$ the message is delayed (2) until the Duellist(i') fights with another Duellist(i'): this occurs eventually (by the inductive hypothesis). The message is then sent to the winner of the fight Duellist($i' + 1$) and, if $i' + 1 < i$ it is again delayed until Duellist($i' + 1$) fights with another Duellist($i' + 1$). This process continues until the message is delayed in a Duellist($i - 1$) waiting for the fight with the other Duellist($i - 1$) (the fight between the two Duellists($i - 1$) takes place by the inductive hypothesis). One of them becomes Duellist(i) and the other becomes Second($i - 1$). If the ATTACK message is waiting at the Duellist, the Lemma is proved; if it is waiting at the Second, it is forwarded to the Duellist(i) because the Second knows how to reach it (by Lemma 4.2).

In case (b), the ATTACK reaches a Second(i') (with $i' < i$), the Second forwards the message to its Duellist($i' + 1$) (4) and the previous proof of case (a) is repeated. □

The next lemma shows that there is exactly one Duellist at level i in each i-cube.

Lemma 4.5 *A Duellist(i) is the only Duellist of an undirected i-cube in which there exist* $2^i - 1$ *Seconds with a level strictly less than* i.

Proof By induction. Each Duellist(0) is a 0-cube. Let the theorem hold for i and consider the fights at level $i+1$. Each Duellist(i) sends an attack through the link labeled $i+1$. From lemma 4.1 (ii) the attack remains inside the $(i+1)$-cube in which there exists a single image Duellist (by inductive hypothesis). From lemma 4.4, the attack reaches this Duellist and only one of the two becomes Duellist($i+1$). $\square$

By Lemmas 4.3, 4.4 and 4.5, it follows that

Theorem 4.3 *The algorithm terminates correctly after* $\log N$ *steps and elects a* leader.

4.2 Election Algorithm with the Distance Model

We present a distributed Election algorithm for hypercubes with a $\Theta(N)$ message complexity using the distance Sense of Direction. We first prove that the symmetry is broken with this model of Sense of Direction and, therefore, that the leader Election does not require a global maximum-finding algorithm.

Theorem 4.4 *For a hypercube with the distance Sense of Direction obtained by a Gray reflected code, exactly four processors have a link labeled* $\frac{N}{2}-1$.

Proof By induction. Let us assume that the Hamiltonian cycle used in the hypercube respects the binary reflected code construction. Without loss of generality, we assume in the following that all the processors have a number in $[0..(N-1)]$ which depends on their respective position in the Hamiltonian cycle.

Clearly, the Theorem is valid for $N = 8$ (as shown in figure 1(b)). By construction, the Hamiltonian cycle for the hypercube with the successive degree is obtained in the following way: assign this smaller hypercube as the first half-cycle, create its reflected image as the second half-cycle, and link each processor and its mirror image.

In the new subset of links obtained by the reflection process (in between the two half-cycles), only the two processors and their respectives images assigned to the nodes at the cycle positions $\frac{N}{4}$ and $\frac{3N}{4}$ positions have a link with a $\frac{N}{2}-1$ distance, thus, only two processors have such a labeled link (the respective images have a link labeled with value $\frac{N}{2}+1$).

In each of the two half-cycles of size $\frac{N}{2}$, by definition, only one link can have a $\frac{N}{2}-1$ distance, since this is the maximum distance inside the half cycle (the outside distance is larger than $\frac{N}{2}$ since it has to count the nodes of the other half cycle). This link always exists since it is the one between the first and the last processor in the half-cycle which are joined (the terminal link of the previous smaller Hamiltonian cycle).

The four processors are in the respective cycle positions: $0, \frac{N}{4}, \frac{N}{2}, 3\frac{N}{4}$ (figure 2(a) where the cycle positions are outside the Hamiltonian cycle). $\square$

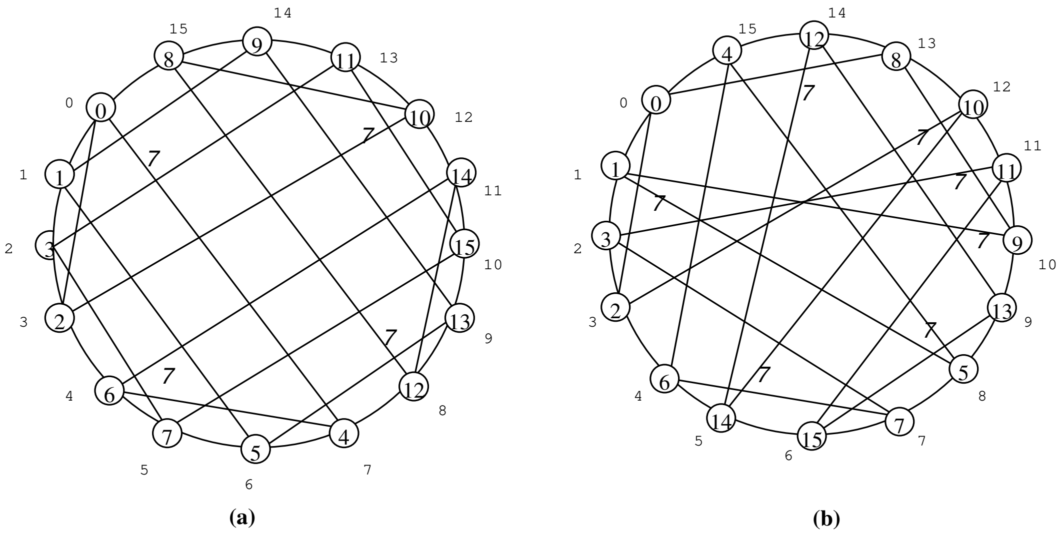


Figure 2: Hamiltonian Cycle Impact on Distance: (a) Reflected, (b) Arbitrary

Remark Note that Theorem 4.4 might not hold if the Gray reflected code is not used. In fact, many other Hamiltonian cycles can be used for building a distance sense of direction in the hypercube providing an arbitrary number of nodes with an edge labeled by a $\frac{N}{2}-1$ distance. Consider, for example, the $\langle 1213414243212343 \rangle$ coordinate cycle, initialized in processor 0 in a 4-dimensional hypercube (figure 2(b)); with this choice of the Hamiltonian cycle seven nodes (instead of four) have an edge labeled $\frac{N}{2}-1$ (in the figure, these nodes have the identities {1,14,5,9,11,10,12} and cycle positions {1,5,8,10,11,12,14,15}).

Corollary 4.1 *For a hypercube with the distance Sense of Direction based on the Gray reflected code, the overall message cost required for a node to detect whether it is elected or not, is* $O(1)$.

Proof The four processors with labels $\frac{N}{2}-1$ and their immediate predecessors build a cycle of 8 processors - the predecessors, and only them, have a link labeled $\frac{N}{2}+1$). An algorithm using only this cycle elects a leader among them in $O(1)$ messages. All the other nodes already know that they are not elected by verifying that none of their links are labeled $\frac{N}{2}-1$ or $\frac{N}{2}+1$. □

Corollary 4.2 *An algorithm with a* $\Theta(N)$ *message complexity for the Election problem can be given for a hypercube oriented with the distance Sense of Direction built on the Gray reflected code.*

Proof A broadcast phase along the cycle which uses exactly N messages is initiated by the Leader and terminates the algorithm (all the processors then know its identity). Each processor receiving the message on link labeled $(N-1)$ forwards it on link labeled 1. The required $\Omega(N)$ bound is reached and, thus, the $\Theta(N)$ message complexity is proved. □

5 From one Sense of Direction to Another

Without a labeling of the links, the message complexity required to build the orientation of hypercubes has been proved to be at least $\frac{1}{2}N(\log N - 1)$, [20].

We show that the message complexity of the orientation labeling decreases if the network is already oriented by another sense of direction. We present two $O(N)$ algorithms to re-label the links from one Sense of Direction to another.

Theorem 5.1 *A hypercube with the dimensional Sense of Direction can be re-oriented with the distance Sense of Direction in $O(N)$ messages.*

Proof The algorithm consists of three phases.

First a unique leader is elected with the algorithm presented in Section 4.1. Second, the Hamiltonian cycle (initiated by the Leader) is built using the binary reflected Gray-code as described in Section 3. During this phase, each processor gets its relative cycle position *cid* on the cycling path (*cid* of initiator is zero). The incoming and outgoing edges respect their order in the coordinate sequence. These links are re-labeled by $N-1$ for the one with the incoming message and by 1 for the other with the outgoing message.

The third phase consists of re-labeling the $n-2$ untouched links, but is computed locally. On each node *cid*, a link previously labeled with the dimension d has to be re-labeled by the distance:

$$(2^d - 1 - 2 \cdot (cid \bmod 2^d)) \bmod N$$

This result is fully based on the fact that the hypercube, by constructing the Hamiltonian cycle, is split (for each d) into a set of equal slices of size 2^d. Each link labeled d joins two reflected images, and, thus, the distance on the cycle between the two adjacent nodes is twice the distance from this node to the end of the cube of size 2^d in the cycle.

The two first phases take $O(N)$ messages respectively, and, thus, we prove the Theorem. □

Theorem 5.2 *A hypercube with the distance Sense of Direction can be re-oriented with the dimensional Sense of Direction in $O(N)$ messages.*

Proof Similar to Theorem 5.1. The algorithm consists of three phases. First a leader is selected as shown in Corollary 4.1 using $O(1)$ messages. Then, the leader initiates a counting message on the Hamiltonian cycle (through link labeled 1), so that each processor is able to get its relative *cid* number. Upon receipt of this message, each processor re-labels its incoming link (previously $N-1$) and its outgoing link (previously 1) regarding the rank on the appropriate coordinate sequence A_n described in Section 3.

The final phase is a local computation which reverses the process used in Theorem 5.1. On each node, a link previously labeled *dist* is re-labeled by the dimension:

$$\lceil \log((dist + 2 \cdot cid) \bmod N) \rceil$$

Only the second phase takes $O(N)$ messages, whereas the first takes only $O(1)$. □

6 Concluding Remarks

Two immediate extensions emerge from these results. First, the convenience of the algorithms presented is emphasized by the fact that the solutions based on Sense of Direction can be extended for bounded degree variation of the hypercube (e.g., the Butterfly, the Cube-Connected-Cycles, the Beneš network, the Shuffle-Exchange, the de Bruijn graph,...). Second, this raises a more general and interesting question: how many labeled links (with sense of direction) are required in a hypercube in order to solve the Election problem with a linear communication complexity ?

After completion of the original paper [2], the authors learned that for the dimensional labeling, linear election algorithms have been independently obtained by Robbins and Robbins [16], and Tel [19]. Tel's algorithm is based on a match-making technique from Mullender and Vitanyi [14], whereas Robbins' algorithm closely resembles our solution. Both use slightly more messages than ours and terminate in $O(N)$ time in the worst-case (the time complexity cannot exceed the message complexity).

Acknowledgements We would like to thank Nicola Santoro for his helpful comments and for bringing this problem to our attention.

References

[1] H. Attiya, J. van Leeuwen, N. Santoro, and S. Zaks. Efficient elections in chordal ring networks. *Algorithmica*, 4:437–446, 1989.

[2] P. Flocchini and B. Mans. Optimal elections in labeled hypercubes. Technical Report TR-93-231, Carleton University, Ottawa, K1S 5B6 Canada, 1993.

[3] P. Flocchini, B. Mans, and N. Santoro. Sense of direction: formal definition and properties. In *Proc. 1994 1st Colloquium on Structural Information and Communication Complexity*, pages 9–34, Ottawa, Canada, 1994.

[4] R.G. Gallager, P.A. Humblet, and P.M. Spira. A distributed algorithm for minimum spanning tree. *ACM Trans. on Progr. Languages and Systems*, 5(1):66–77, 1983.

[5] E.N. Gilbert. Gray codes and paths on the n-cube. *Bell Systems Technical Journal*, 37:815–826, 1958.

[6] F. Harary, J.P. Hayes, and H.-J. Wu. A survey of the theory of hypercubes graphs. *Comput. Math. Applic.*, 15(4):277–289, 1988.

[7] Z. Harpaz and S. Zaks. Sense of direction in complete distributed networks. Technical Report TR-453, Department of Computer Science, Technion, Israel, April 1987. See also 23rd Allerton Conference on Communication, Control and Computing, 1985.

[8] T.Z. Kalamboukis and S.L. Mantzaris. Towards optimal distributed election on chordal rings. *Information Processing Letters*, 38:265–270, 1991.

[9] E. Korach, S. Moran, and S. Zaks. Tight lower and upper bounds for a class of distributed algorithms for a complete network of processors. In *Proc. 1984 Symposium on Principles of Distributed Computing*, pages 199–207, Vancouver, 1984.

[10] E. Korach, S. Moran, and S. Zaks. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theoretical Computer Science*, 64:125–132, 1989.

[11] F. Thomson Leighton. *Introduction to parallel algorithms and architectures; arrays, trees, hypercubes*. M.I.T. Press, 1992.

[12] M.C. Loui, T.A. Matsushita, and D.B. West. Election in complete networks with a sense of direction. *Information Processing Letters*, 22:185–187, 1986. See also *Information Processing Letters*, 28:327, 1988.

[13] B. Mans and N. Santoro. On the impact of sense of direction in arbitrary networks. In *Proc. 1994 IEEE 14th International Conference on Distributed Computing Systems (ICDCS'94)*, pages 258–265, Poznan, Poland, June 21-24 1994.

[14] S.J. Mullender and P.M. Vitanyi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

[15] G.L. Peterson. Efficient algorithms for elections in meshes and complete networks. Technical Report 140, Computer Science, Univ. of Rochester, NY-14627, 1985.

[16] S. Robbins and K.A. Robbins. Choosing a leader on a hypercube. In *Proc. 1990 International conference on databases, parallel architectures and their applications (PARBASE'90)*, pages 469–471, Miami Beach, USA, 1990.

[17] N. Santoro. On the message complexity of distributed problems. *Journal of Computing Information Science*, 13:131–147, 1984.

[18] N. Santoro. Sense of direction, topological awareness and communication complexity. *SIGACT NEWS*, 2(16):50–56, summer 1984.

[19] G. Tel. Linear election for oriented hypercube. Technical Report TR-RUU-CS-93-39, Utrecht University, Department of Computer Science, The Netherlands, 1993.

[20] G. Tel. Network orientation. *International Journal of Foundations of Computer Science*, 5(1):23–57, 1994.