Linear Election for Oriented Hypercubes

Gerard Tel*
Department of Computer Science, University of Utrecht,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.
Email: gerard@cs.ruu.nl

Abstract

In this article we propose an election algorithm for the oriented hypercube, where each edge is assumed to be labeled with its dimension in the hypercube. The algorithm exchanges $O(N)$ messages and uses $O(\log^2 N)$ time (where N is the size of the cube).

A randomized version of the algorithm achieves the same (expected) message and time bounds, but uses messages of only $O(\log \log N)$ bits and can be used in anonymous hypercubes.

1 Introduction

The election problem is one of the most intensively studied problems in distributed algorithms research. In addition to its practical importance, the problem has developed to a "bench-mark" to study the complexity effects of different model assumptions. In one line of this research, initiated by Santoro [San84], it was investigated how knowledge of topology or orientation influences the complexity of the election problem. (In the following discussion we only consider message complexity, and N and E denote the number of processes and links in the network.) For networks of arbitrary, unknown topology an $O(N \log N + E)$ algorithm was given by Gallager *et al.* [GHS83], and this is also a lower bound.

The existence of an orientation does not help in two important classes of networks, namely rings and tori. In unoriented rings, Franklin's algorithm [Fra82] uses $O(N \log N)$ messages, while the $\Omega(N \log N)$ lowerbound [Bur80, Bod91] applies to oriented rings as well. Peterson [Pet85] proposed an $O(N)$ algorithm for election in unoriented tori; clearly $\Omega(N)$ messages are needed in oriented tori as well.

For another important class of networks, namely cliques, orientation does help. While an $\Omega(N \log N)$ lower bound on election in unoriented cliques was shown by Korach *et al.* [KMZ84], an $O(N)$ algorithm for oriented cliques was given by Loui *et al.* [LMW86].

The question whether orientation helps in hypercubes has remained open. Using an efficient traversal possible in oriented hypercubes, and the construction of Korach *et al.* [KKM90], election can be performed in $2N \log N$ messages in the oriented hypercube. But, disappointingly, because the hypercube has only $O(N \log N)$ edges, an $O(N \log N)$ complexity is also achieved when the standard algorithm of Gallager *et al.* [GHS83] is

*The work of this author was supported by ESPRIT Basic Research Action No. 7141 (project ALCOM II: *Algorithms and Complexity*). and by the Netherlands Organization for Scientific Research (NWO) under contract NF 62-376 (NFI project ALADDIN: *Algorithmic Aspects of Parallel and Distributed Systems*).

applied. For neither the oriented, nor the unoriented case was the $O(N \log N)$ solution known to be optimal.

In this paper we present an election algorithm for the oriented hypercube that uses $O(N)$ messages, showing that the known algorithms can be improved for the oriented case. It is commonly believed that for the unoriented case the $O(N \log N)$ solution is optimal; if this belief is correct, our result shows that orientation helps in hypercubes. Our paper is organized as follows. Section 2 describes the structure of the hypercube and the orientation. Section 3 presents our algorithm. Section 4 gives the randomized version of the algorithm. Section 5 summarizes the achievements and lists open problems.

2 Model Description

The n-dimensional hypercube is a graph on $N = 2^n$ nodes, which, due to its attractive combinatorial properties, is a popular choice for the topology of multiprocessor computers.

Definition 2.1 *A graph G on $N = 2^n$ nodes is a* hypercube *if the nodes can be labeled with the N different sequences of n bits, such that an edge between two nodes exists if and only if the labels of the nodes differ in exactly one bit.*

If G is a hypercube, the labeling as referred to in the definition is not unique; there exist $N \cdot n!$ different label assignments with this property. In the remainder of this paper, fix one such labeling and refer to it as the *canonical node labeling*; we shall refer to nodes using their canonical label. The canonical label of node u is denoted $u_0 u_1 ... u_{n-1}$, abbreviated $\vec{u}$. The *direction* of the edge between $\vec{u}$ and $\vec{v}$ is the (unique) index i such that $u_i \neq v_i$.

Properties of hypercube graphs. Many graph-theoretic results about hypercubes are known; see, e.g., Saad and Schultz [SS88]. The hypercube is regular of degree n, and the distance between two nodes equals the *Hamming distance* between their canonical labels. The diameter and the radius of the hypercube are n. The number of edges in the hypercube is $\frac{1}{2}nN$.

The hypercube has many induced subgraphs that are hypercubes of smaller dimension; given two nodes u and v at distance d, the set of nodes $\{w : d(w, u) + d(w, v) = d(u, v)\}$ induces a subgraph that is a hypercube of dimension d. This subgraph is referred to as the *face* spanned by u and v and $\vec{w}$ is in the face iff for every direction i, $w_i = u_i$ or $w_i = v_i$.

The hypercube graphs have an elegant recursive structure. To construct a labeled $(d+1)$-dimensional hypercube, take to d-dimensional hypercube and extend all labels in one hypercube with a 0 and all labels in the other hypercube with a 1. For each label $\vec{u}$ of d bits, add an edge (of direction d) between $\vec{u}0$ and $\vec{u}1$.

Hypercube networks and election. We consider processor networks of the hypercube topology, i.e., networks whose interconnection topology is a hypercube graph. An *election algorithm* is a program that is executed by every process and brings the network in a configuration where there is exactly one process in a special state *leader*, while all other processes are in a state *non-leader*. Note that the program to be executed is the same for every process.

If the processes know their canonical label, the election problem has a trivial solution: each process enters state *leader* if its label equals $\vec{0}$ and *non-leader* otherwise. This program

is correct because there is exactly one process with label $\vec{0}$. We assume from now on that the processes do not know their canonical label, but nodes are distinguished by arbitrarily chosen names that have no topological significance. (The case that processes are *anonymous*, i.e., are not distinguished at all, is discussed in Section 4.)

A distinction can be made between *unoriented* and *oriented* hypercubes. In unoriented networks, a process distinguishes between its links by using different, but uninterpreted names. In the oriented hypercube, a process knows the direction of each of its adjacent links. It is believed that the availability of an orientation helps to reduce the communication complexity of computations in the hypercube. A network traversal [KKM90], starting from an arbitrary process, can be performed using N messages in an oriented hypercube, while the best known algorithm for unoriented hypercubes takes $N \log N$ messages.

The election algorithm by Gallager *et al.* [GHS83] can be applied to unoriented hypercubes and uses $O(N \log N)$ messages in this case; this is the best known for unoriented hypercubes. Our algorithm for oriented hypercubes uses only $O(N)$ messages, thus supporting the belief that orientation helps to reduce the complexity of election in hypercubes. It should be mentioned, though, that the $O(N \log N)$ algorithm for unoriented hypercubes was not yet shown to be optimal.

3 The Election Algorithm

The election algorithm heavily relies on the recursive structure of the hypercube graph. A hypercube of dimension $d+1$ consists of two copies of the hypercube of dimension d, where each pair of corresponding nodes is connected with an edge of direction d. To elect a leader in this hypercube, the algorithm first (recursively) elects a leader in both of the constituting hypercubes of dimension d, and then proceeds to elect one of the two leader processes. To avoid confusion between leadership at different levels of recursion, a process is called a *d-leader* if it has won the election in a d-dimensional hypercube. The base case of this algorithm, election in a hypercube of dimension 0, is easy; the network consists of exactly one node, which becomes a 0-leader immediately.

3.1 The Tournament: Overview

After the election of a d-leader in the two d-dimensional sub-hypercubes, the processes must cooperate to elect exactly one of these to become the $(d+1)$-leader. A tournament between two processes that can communicate directly is easily organized. Each process sends the other one a message containing its name; the process receiving a larger name than its own becomes *non-leader*, the process receiving a smaller name than its own becomes *leader*.

The tournament between the two d-leaders is organized in the same way, but with the difficulty that the processes do not know how to reach the leader in the other, or even their own, sub-hypercube. As a first step, process p that becomes d-leader sends a tournament message $\langle \textbf{tour}, p, d \rangle$, containing its name and the phase number d, through its edge in direction d. It is the responsibility of the receiving node, called the *entry* node, to forward this message to its d-leader; see Fig. 1.

It remains to forward the message to the d-leader in an efficient way; observe that the relative position of the d-leader and the entry node can be arbitrary and is not known to the entry node. It it too expensive for the d-leader to announce its position to all nodes in

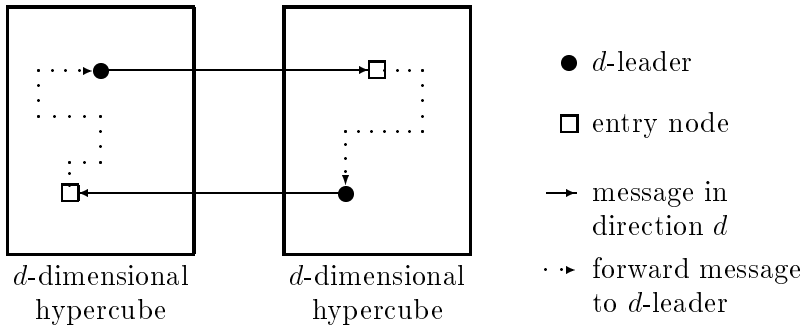

Figure 1: Message forwarding in the tournament.

the d-cube so that the entry node can forward the message in d steps. This announcement would cost $2^d - 1$ messages, leading to an $O(N \log N)$ overall complexity of the election. Similarly, it is too expensive to have to entry node broadcast the tournament message through the d-cube; this would also cost $2^d - 1$ messages.

3.2 The Match-Making Technique

The forwarding of the message can be seen as a *match-making* problem (see Mullender and Vitanyi [MV88]) and can be solved using $O(\sqrt{2^d})$ messages and in $d + 1$ time (which is optimal).

To make a match between the d-leader and the entry node, the d-leader announces its leadership to all nodes in a $\lfloor d/2 \rfloor$-dimensional face, referred to as the leader's *row*. The entry node broadcasts the tournament message through a $\lceil d/2 \rceil$-dimensional face called its *column*. As each row intersects each column in exactly one process (as will be shown below), there is one process, called the *match process*, that receives both the announcement from the d-leader and the tournament message. The match process forwards the tournament message further to the d-leader via the spanning tree induced by the announcement messages.

Definition 3.1 *Consider the hypercube of dimension d.*
The row *with index $u_{\lfloor d/2 \rfloor}..u_{d-1}$ is the subset of nodes $\{x_0..x_{\lfloor d/2 \rfloor - 1} u_{\lfloor d/2 \rfloor}..u_{d-1}\}$.*
The column *with index $u_0..u_{\lfloor d/2 \rfloor} - 1$ is the subset of nodes $\{u_0..u_{\lfloor d/2 \rfloor - 1} x_{\lfloor d/2 \rfloor}..x_{d-1}\}$.*

Lemma 3.2 *Each node belongs to exactly one row and to exactly one column. Any row intersects any column in exactly one process.*

Proof. Node $\vec{u} = u_0..u_{\lfloor d/2 \rfloor - 1} u_{\lfloor d/2 \rfloor}..u_{d-1}$ belongs to the row with index $u_{\lfloor d/2 \rfloor}..u_{d-1}$ but not to any other row. This node belongs to the column with index $u_0..u_{\lfloor d/2 \rfloor - 1}$.

Row $u_{\lfloor d/2 \rfloor}..u_{d-1}$ and column $u_0..u_{\lfloor d/2 \rfloor - 1}$ intersect in process $u_0..u_{\lfloor d/2 \rfloor - 1} u_{\lfloor d/2 \rfloor}..u_{d-1}$, which is the only process that belongs to both subsets. □

Algorithm 2 shows how to broadcast the $\langle \mathbf{ann}, d \rangle$ message through a row using only $2^{\lfloor d/2 \rfloor} - 1$ messages, and without using the canonical node labels. Each process stores the link through which the message was received (variable $fath_p$), thus building a spanning tree of the row.

var $fath_p : -1...d-1$;

To initiate a broadcast:
begin $fath_p := -1$; $broad(\lfloor d/2 \rfloor, d)$ **end**

Upon receiving $\langle \mathbf{ann}, d \rangle$ via link i:
begin $fath_p := i$; $broad(i, d)$ **end**

procedure $broad(i)$:
begin if $i > 0$
then send $\langle \mathbf{ann}, d \rangle$ through link $i-1$; $broad(i-1, d)$ **end**
end

Algorithm 2: Broadcasting in a row.

Lemma 3.3 *Alg. 2 broadcasts the message* $\langle \mathbf{ann}, d \rangle$ *through a row using* $2^{\lfloor d/2 \rfloor} - 1$ *messages, and builds a spanning tree of the row, rooted at the initiator, of depth* $\lfloor d/2 \rfloor$.

Proof. We first show the following by induction on i: if $broad(i, d)$ is executed by process $\vec{u} = u_0..u_{d-1}$, the message $\langle \mathbf{ann}, d \rangle$ is received exactly once by the processes in $\{x_0..x_{i-1}u_i..u_{d-1}\} \setminus \{\vec{u}\}$, and a spanning tree on these nodes is built of depth i.

Case $i = 0$**:** Execution of $broad(0, d)$ is a skip, so no process will receive anything; indeed, $\{u_0..u_{d-1}\} \setminus \{\vec{u}\}$ is the empty set.

Case $i + 1$**:** Execution of $broad(i+1, d)$ by $\vec{u}$ first sends an $\langle \mathbf{ann}, d \rangle$ message via link i, that is, to node $\vec{u'} = u_0..u_{i-1}\bar{u}_i u_{i+1}..u_d$. ($\bar{u}_i$ is the complement of u_i.) By induction, the subsequent execution of $broad(i, d)$ in u and u' (the latter upon receipt of the $\langle \mathbf{ann}, d \rangle$ message from u) results in all processes in

$$\{x_0..x_{i-1}u_iu_{i+1}..u_{d-1}\} \setminus \{\vec{u}\} \quad \text{and} \quad \{x_0..x_{i-1}\bar{u}_iu_{i+1}..u_{d-1}\} \setminus \{\vec{u'}\}$$

receiving the message once. Consequently, all processes in

$$\{x_0..x_{i-1}x_iu_{i+1}..u_{d-1}\} \setminus \{\vec{u}\}$$

receive the message exactly once.

The recursive calls both build a spanning tree of depth i, but one is rooted at u' and hence become hooked in at depth 1, which brings the overall depth at $i + 1$.

Thus, the initialization of a broadcast causes all processes in the initiator's row to receive the message exactly once. The message complexity is $2^{\lfloor d/2 \rfloor} - 1$ because all processes in the row except one receive $\langle \mathbf{ann}, d \rangle$ once. □

A similar procedure is used to broadcast the tournament message through the column of the entry node. This broadcast takes $2^{\lceil d/2 \rceil} - 1$ messages.

The tournament between the two d-leaders is organized as follows.

1. A d-leader p sends a $\langle \mathbf{tour}, p, d \rangle$ message via link d.
2. A d-leader announces its leadership in its row by calling $broad(\lfloor d/2 \rfloor, d)$.

3. An entry node (i.e., a node receiving $\langle\, \mathbf{tour}, p, d \,\rangle$ via link d) broadcasts the node through its column.
4. A non-d-leader q in the row of the leader (i.e., q has received an $\langle\, \mathbf{ann}, d \,\rangle$ message) receiving a $\langle\, \mathbf{tour}, p, d \,\rangle$ message sends the message to $fath_q$.
5. A d-leader q receiving a $\langle\, \mathbf{tour}, p, d \,\rangle$ message compares p with q; if $q > p$, q becomes $(d+1)$-leader, and q becomes non-leader otherwise.

3.3 Complexity Analysis

The message complexity. Let $T(d)$ be the number of messages exchanged in a tournament between two d-leaders and $E(n)$ the number of messages used by the election algorithm (on a hypercube of dimension n). As an election requires two elections on smaller cubes and one tournament, we find the recursion

$$E(n) = \begin{cases} 0 & \text{if } n = 0 \\ 2 \cdot E(n-1) + T(n-1) & \text{otherwise.} \end{cases}$$

In the analysis of the message complexity we need the well-known sum of the infinite geometric series (assuming $\alpha < 1$) and its derivative to α:

$$\sum_{d=0}^{\infty} \alpha^d = \frac{1}{1-\alpha} \qquad \text{and} \qquad \sum_{d=0}^{\infty} d \cdot \alpha^{d-1} = \frac{1}{(1-\alpha)^2}.$$

Counting the number of messages exchanged in each of the steps of the tournament between two d-leaders, we find:

Step 1: 2 messages.
Step 2: $2 \cdot (2^{\lfloor d/2 \rfloor} - 1)$ messages.
Step 3: $2 \cdot (2^{\lceil d/2 \rceil} - 1)$ messages.
Step 4: at most $2 \cdot \lfloor d/2 \rfloor$ messages.

Summing up, find $T(d) \leq 2 \cdot (2^{\lfloor d/2 \rfloor} + 2^{\lceil d/2 \rceil} + \lfloor d/2 \rfloor - 1)$.

For even d we have $\lfloor d/2 \rfloor = \lceil d/2 \rceil = d/2$, and hence $T(d) \leq 4 \cdot 2^{d/2} + d - 2$.

For odd d we have $2^{\lfloor d/2 \rfloor} = \frac{1}{2}\sqrt{2} \cdot 2^{d/2}$ and $2^{\lceil d/2 \rceil} = \sqrt{2} \cdot 2^{d/2}$. This implies $T(d) \leq (3\sqrt{2})2^{d/2} + d - 3$.

As $3\sqrt{2} > 4$, the relation $T(d) < (3\sqrt{2})2^{d/2} + d - 2$ holds both for even and odd d.

Subsequently we write $F(n) = E(n)/2^n$, and using the recursion for E we find

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ F(n-1) + \dfrac{T(n-1)}{2^n} & \text{otherwise,} \end{cases}$$

which allows to write $F(n)$ as a sum and bound it by an infinite series:

$$F(n) = \sum_{d=0}^{n-1} \frac{T(d)}{2^{d+1}} \quad \leq \quad \sum_{d=0}^{\infty} \frac{T(d)}{2^{d+1}}.$$

Subsequently we use the relation $T(d) < (3\sqrt{2})2^{d/2} + d - 2$ to bound $F(n)$ as follows

$$\begin{aligned}
\sum_{d=0}^{\infty} \frac{T(d)}{2^{d+1}} &< \sum_{d=0}^{\infty} \frac{(3\sqrt{2})2^{d/2}}{2^{d+1}} &+ \sum_{d=0}^{\infty} \frac{d}{2^{d+1}} &- \sum_{d=0}^{\infty} \frac{2}{2^{d+1}} \\
&= \frac{3\sqrt{2}}{2} \cdot \sum_{d=0}^{\infty} (\frac{1}{2}\sqrt{2})^d &+ \frac{1}{4} \cdot \sum_{d=0}^{\infty} d \cdot (\frac{1}{2})^{d-1} &- \sum_{d=0}^{\infty} (\frac{1}{2})^d \\
&= \frac{3\sqrt{2}}{2} \cdot \frac{1}{1 - \frac{1}{2}\sqrt{2}} &+ \frac{1}{4} \cdot \frac{1}{(1 - \frac{1}{2})^2} &- 2 \\
&= 3\sqrt{2} + 2 \quad \approx \quad 6.24.
\end{aligned}$$

It follows that $E(n) < 6.24 \cdot N$.

The constant can be reduced by carrying out a more precise analysis, where the sum $F(n)$ is calculated for even and odd d separately.

Time complexity. It will first be assumed that all processes initiate the algorithm independently, and the latest process does so at time t_0; we shall show that a leader is elected at time $t_0 + O(n^2)$.

Assume all d-leaders are elected at time t_d. At time $t_d + \lfloor d/2 \rfloor$ the announcement of leadership in the row has completed. At time $t_d + 1$ the entry nodes have received the tournament messages, and the broadcast of this message in the column is completed at time $t_d + 1 + \lceil d/2 \rceil$. Hence the forwarding of the messages through the row starts at the latest at time $t_d + \max(\lfloor d/2 \rfloor, \lceil d/2 \rceil + 1)$, and the d-leaders receives the message time $t_{d+1} \leq t_d + \max(\lfloor d/2 \rfloor, \lceil d/2 \rceil + 1) + \lfloor d/2 \rfloor = t_d + (d + 1)$.

The time of election of the n-leader is then bounded by

$$t_n = t_0 + \sum_{d=0}^{n-1} (d + 1) = t_0 + \frac{1}{2} n(n - 1),$$

hence the algorithm elects within $\frac{1}{2} \log^2 N$ time.

Initiation by a Subset of Processes. The algorithm can also be initiated by any (non-empty) subset of the processes, but the time complexity is linear in this case. A process starts the algorithm either spontaneously or on receipt of the first message of the algorithm.

Lemma 3.4 *If any subset of processes initiates the algorithm spontaneously, all processes will eventually start executing the algorithm and it terminates in $O(N)$ time.*

Proof. Let p be a process that initiates the algorithm. We show by induction on d that all nodes in the d-dimensional subcube to which p belongs eventually join the algorithm.

Case $d = 0$**:** The 0-dimensional cube only contains p, and p is a starter.

Case $d + 1$**:** By induction, all processes in p's d-dimensional cube start the algorithm, hence they elect one d-leader (not necessarily p), which sends a tournament message

in direction d. The entry node receiving this message starts the algorithm in the other d-dimensional cube, and by induction all processes in that d-cube eventually start the algorithm, too. So all processes in p's $(d+1)$-dimensional subcube eventually start the algorithm.

The time bound holds because $O(N)$ messages are exchanged by the algorithm, and the time complexity does not exceed the message complexity. □

3.4 Proper Termination of the Algorithm

After election of the final n-leader there are still processes in the hypercube waiting to receive messages. Indeed, all processes in a d-leader's row are still ready to receive and forward a tournament message. All processes in an entry node's column are still ready to receive an announce message and forward the tournament message to the leader.

After its election, the n-leader can announce its election to all nodes in the hypercube, causing them to terminate and purge all pending messages. This costs $N-1$ messages.

4 A Randomized Algorithm for Anonymous Hypercubes

The node names are only used for breaking the symmetry between nodes on a two-by-two basis. Randomly selected strings can be used for this purpose, too, and this yields a randomized algorithm with an even lower bit-complexity that can also be used in an anonymous hypercube.

When becoming d-leader, a process randomly selects a string of k bits and includes it in the tournament message instead of its name. If the two competing d-leaders selected different strings (this has probability $1-2^{-k}$), the one with larger string becomes leader. If the two d-leaders selected the same string (a *clash* occurs), both select a new random string and send a new tournament message. As the probability of a clash is only 2^{-k}, the expected number of tournament message exchanges in any single tournament rises from 1 to $\frac{1}{1-2^{-k}}$, that is, remains practically unchanged.

The number of bits per message is $k + \log\log N$ (because messages also contain the recursion level), so it is reasonable to set $k = \log\log N$. We obtain a randomized algorithm using $O(N)$ messages, $O(N \log\log N)$ bits, $O(\log^2 N)$ time.

5 Conclusions, Open Problems, and Related Work

We proposed a linear election algorithm for oriented hypercubes. The algorithm exchanges no more than $6.24...\cdot N$ messages, each containing at most one process name and an integer of $\log\log N$ bits. A randomized version of the algorithm was also presented; this algorithm can be used in anonymous hypercubes and its messages contain only $O(\log\log N)$ bits.

Open questions that remain are:

- Prove that the best known election algorithm for unoriented hypercubes, with a complexity of $O(N \log N)$ messages, is optimal, or give a better algorithm.
- Prove an $\Omega(N \log N)$ lower bound or an $o(N \log N)$ upper bound on the complexity of traversing an unoriented hypercube.

Comparison with [FM93]. An election algorithm with linear complexity was independently found by Flocchini and Mans [FM93]. Like our algorithm, this algorithm exploits the recursive structure of the hypercube to successively elect leaders in higher and higher dimensional sub-hypercubes. Instead of our match-making using rows and columns, each non-leader process forwards a tournament message (via a shortest path) to the process by which it was defeated at a lower level of recursion.

Our tournament uses more messages than the tournament implemented by Flocchini and Mans (*viz.*, $O(\sqrt{N})$ versus $O(\log^2 N)$). It turns out, however, that the complexity of the tournament is insignificant for the overall complexity of the election (as long as it is bounded by $O(N^{1-\epsilon})$ for some positive ϵ). Consequently, the message complexities of the two algorithms are equal.

Our algorithm is better than the algorithm of Flocchini and Mans with respect to time and bit complexity. Because we pass messages in parallel during the tournament, the time complexity of the tournament is bounded by $\log N$, which results in an overall time complexity of $O(\log^2 N)$. The messages in the tournaments are passed serially in [FM93], which results in a $\Theta(\log^2 N)$ time per tournament, and an $\Theta(\log^3 N)$ overall time complexity. Moreover, the messages in [FM93] may contain a process name plus $\log^2 N$ bits, while our messages contain only a name plus $\log \log N$ bits (and $O(\log \log N)$ bits in the randomized version).

References

[Bod91] Bodlaender, H. L. New lower bound techniques for distributed leader finding and other problems on rings of processors. *Theoretical Comput. Sci.* **81** (1991), 237–256.

[Bur80] Burns, J. E. A formal model for message passing systems. Tech. Rep. TR–91, Indiana University, 1980.

[FM93] Flocchini, P., and Mans, B. Optimal elections in labeled hypercubes. Tech. rep., Carleton School of Computer Science, Ottawa, 1993.

[Fra82] Franklin, W. R. On an improved algorithm for decentralized extrema finding in circular configurations of processors. *Commun. ACM* **25**, 5 (1982), 336–337.

[GHS83] Gallager, R. G., Humblet, P. A., and Spira, P. M. A distributed algorithm for minimum weight spanning trees. *ACM Trans. Program. Lang. Syst.* **5** (1983), 67–77.

[KKM90] Korach, E., Kutten, S., and Moran, S. A modular technique for the design of efficient leader finding algorithms. *ACM Trans. Program. Lang. Syst.* **12** (1990), 84–101.

[KMZ84] Korach, E., Moran, S., and Zaks, S. Tight upper and lower bounds for some distributed algorithms for a complete network of processors. In *Symp. on Principles of Distributed Computing* (1984), pp. 199–207.

[LMW86] Loui, M. C., Matsushita, T. A., and West, D. B. Election in a complete network with a sense of direction. *Inf. Proc. Lett.* **22** (1986), 185–187. *Addendum:* Inf. Proc. Lett. 28:327, 1988.

[MV88] Mullender, S. J., and Vitanyi, P. M. B. Distributed match-making. *Algorithmica* **3** (1988), 367–391.

[Pet85] Peterson, G. L. Efficient algorithms for elections in meshes and complete networks. Tech. Rep. TR 140, Dept. of Computer Science, Univ. of Rochester, Rochester, NY 14627, 1985.

[San84] Santoro, N. Sense of direction, topological awareness, and communication complexity. *ACM SIGACT News* **16** (1984), 50–56.

[SS88] Saad, Y., and Schultz, M. H. Topological properties of hypercubes. *IEEE Trans. Comput.* **C–37**, 7 (1988), 867–872.

Contents