0(n log n) LOWER AND UPPER BOUNDS FOR A CLASS OF DISTRIBUTED ALGORITHMS FOR A COMPLETE NETWORK OF PROCESSORS (Extended Abstract)
by
E. Korach*, S. Moran** and S. Zaks**
Technical Report #272
May 1983

* IBM - Scientific Center, Technion City, Haifa, Israel
** Computer Science Dept., Technion, Haifa, Israel
1. INTRODUCTION

The model under investigation is a network of \( n \) processors with distinct identities \( \text{identity}(1), \text{identity}(2), \ldots, \text{identity}(n) \). No processor knows any other processor's identity. Each processor has some communication lines, connecting him to some others. The processor knows the lines connected to himself, but not the identities of his neighbors. The communication is done by sending messages along the communication lines. The processors all perform the same algorithm, that includes operations of (1) sending a message to a neighbor, (2) receiving a message from a neighbor and (3) processing information in their (local) memory.

We assume that the messages arrive, with no error, in a finite time, and are kept in a queue until processed. We also assume that any non-empty set of processors may start the algorithm; a processor that is not a starter remains asleep until a message reaches him.

The communication network is viewed as an undirected graph \( G = (V,E) \) with \( |V| = n \), and we assume that the graph \( G \) is connected. We refer to algorithms for a given network as algorithms acting on the underlying graph.

Working within this model, when no processor knows the value of \( n \), a spanning tree is found in [4] in \( O(n \log n + |E|) \) messages for a general graph. A leader in a network is found in [3], where \( n \) is known to every processor, in an expected number of messages which is \( O(n \log n) \) (independent of \(|E|\)), and the worst case is not analyzed (but is said to be \( O(n |E|) \)).

\( O(n \log n) \) lower and upper bounds for the problem of distributively finding a leader in a circular network of processors are known, see [1,6] for the lower bound and [2,4,5,7] for the upper bound.

We address the class of algorithms for complete graphs, that must use edges of a spanning subgraph in every possible execution. The problems of choosing a leader, finding a maximum and finding a spanning tree clearly require algorithms in this class.
We first prove lower bounds of \(O(n \log n)\) for the number of edges (hence messages) used by any such algorithm. In the full paper we extend the technique developed here to prove a lower bound of \(O(n^2)\) edges for another class of algorithms for complete graphs; these include algorithms for constructing a perfect matching and constructing a Hamiltonian path (or cycle).

Next we present and discuss an algorithm that attains this bound of \(O(n \log n)\) messages for the problem of choosing a leader in a complete graph. This algorithm can be used for optimally solving (up to a constant factor) other problems in this class, among which are the problems of finding the maximum and constructing a spanning tree. The proofs for the correctness and for the complexity of this algorithm are given in the full paper.

Our results heavily use the fact that the underlying graph is complete, which enables us to use, in the worst case, a number of messages that is much smaller than the number of edges \((O(n \log n)\ vs. \ O(n^2))\). This property is not shared by any of the results in [1] - [7]; in fact, we show in the full paper that \(O(|E|)\) messages are required for similar algorithms on some general classes of graphs (including the class of graphs that become complete after deleting one node).

2. LOWER BOUNDS FOR GLOBAL ALGORITHMS

In this section we study lower bounds for global algorithms (to be defined shortly). We first need some definitions.

Let \(A\) be a distributed algorithm acting on a graph \(G = (V,E)\). An execution of \(A\) consists of events, each being either sending a message, receiving a message or doing some local computation. Without loss of generality, we may assume that during every execution no two messages are sent in exactly the same time. Therefore, with each execution we can associate a sequence \(\text{SEND} = (send_1, send_2, \ldots, send_k)\) that includes all the events of the first type in their order of occurrence (if there are no such events then \(\text{SEND}\) is the empty sequence). Each event \(send_i\) we identify with the pair \((v(send_i), e(send_i))\).
where $v(send_i)$ is the node sending the message and $e(send_i)$ is the edge used by it.

Let $SEND(t)$ be the prefix of length $t$ of the sequence $SEND$, namely $SEND(t) = <send_1, ..., send_t>$ ($SEND(0)$ is the empty sequence). If $t < t'$ then we say that $SEND(t')$ is an extension of $SEND(t)$ and we denote $SEND(t) < SEND(t')$ $SEND$ is called a completion of $SEND(t)$. Note that a completion is not necessarily unique.

Let $NEW = NEW(SEND)$ be the subsequence $<new_1, new_2, ..., new_t>$ of the sequence $SEND$ that consists of all the events in $SEND$ that use previously unused edges. (An edge is used if a message has been already sent along it.) This means that the message $send_i = (v(send_i), e(send_i))$ belongs to $NEW$ if and only if $e(send_i) 
eq e(send_j)$ for all $j < i$. $NEW(t)$ denotes the prefix of size $t$ of the sequence $NEW$.

Define the graph $G(NEW(t)) = (V,E(NEW(t)))$ where $E(NEW(t))$ is the set of edges used in $NEW(t)$, and call it the graph covered by the sequence $NEW(t)$. If for every execution of the algorithm $A$ the corresponding graph $G(NEW)$ is connected then we term this algorithm global.

The edge complexity $e(A)$ of an algorithm $A$ acting on a graph $G$ is the maximal length of a sequence $NEW$ over all executions of $A$.

The message complexity $m(A)$ of an algorithm $A$ acting on a graph $G$ is the maximal length of a sequence $SEND$ over all executions of $A$. Clearly $m(A) \geq e(A)$.

For each algorithm $A$ and graph $G$ we define the exhaustive set of $A$ with respect to $G$, denoted by $EX(A,G)$ (or $EX(A)$ when $G$ is clear from the context), as the set of all the sequences $NEW(t)$ corresponding to possible executions of $A$.

By the properties of distributed algorithms the following facts hold for every algorithm $A$ and every graph $G$:

1. the empty sequence is in $EX(A,G)$.
2. if two sequences $NEW_1$ and $NEW_2$, which do not interfere with each other, are
in $EX(A,G)$, then so is also their concatenation $NEW_1; NEW_2$. ($NEW_1$ and $NEW_2$ do not interfere if no two edges $e_1$ in $NEW_1$ and $e_2$ in $NEW_2$ have a common end point; this means that the corresponding partial executions of $A$ do not affect each other and can, in fact, be merged in any specified order.)

If $NEW(t)$ is a sequence in $EX(A,G)$ with a last element $(v,e)$, and if $e'$ is an unused edge adjacent to $v$, then the sequence obtained from $NEW(t)$ by replacing $e$ by $e'$ is also in $EX(A,G)$. (This reflects the fact that a node cannot distinguish between his unused edges.)

Note that, these three facts do not imply that $EX(A,G)$ contains any non-empty sequence.

If the algorithm $A$ is global then the following fact holds as well:

If $NEW(t)$ is in $EX(A,G)$ and $C$ is a proper component of $G(NEW(t))$, then there is an extension of $NEW(t)$ in which the first new message $(v,e)$ satisfies $v \in C$. (This reflects the facts that some unused edge will eventually carry a message and that arbitrarily long delays can be imposed on the nodes not in $C$.)

**Theorem.** Let $A$ be a global algorithm acting on a complete graph $G$ with $n$ nodes. Then the edge complexity $e(A)$ of $A$ is at least $O(n \log n)$.

**Sketch of proof:** For $1 \leq k < n$ let $t(k)$ be the minimal number of edges required by $A$, in the worst case, to cover a connected subgraph of $G$ with $k$ nodes. Using facts $f, 2 - f, 4$ it is shown that $t(2k+1) \geq 2t(k) + k + 1$

which implies the theorem.

From this theorem it follows that

**Theorem:** Let $A$ be a global algorithm acting on a complete graph $G$ with $n$ nodes. Then the message complexity $m(A)$ of $A$ is at least $O(n \log n)$.

### 3. The Algorithm

We now present and discuss the $O(n \log n)$ distributed algorithm for choosing a leader in a complete network of processors.
Each node in the network has a *state*, that is either *KING* or *CITIZEN*. Initially every node is a king (i.e. *state* = *KING*), and - except for one - every one will eventually become a citizen (a citizen will never become a king again).

Each king is a root of a directed tree which is his kingdom. All the other nodes of this tree are citizens of this kingdom, and each node knows his father and sons. Each node \( i \) also stores the identity \( k(i) \) and the *phase* \( \phi(i) \) of his king, which are updated during the execution of the algorithm. At the beginning of the algorithm \( k(i) = \text{identity}(i) \) and \( \phi(i) = 0 \) for each \( i \).

A king is trying to increase his kingdom by sending messages towards other kings (possibly through their citizens), asking them to join, together with their kingdoms, his kingdom.

A citizen, upon receiving messages, can transfer them to (or from) his king along the tree edges, delay them or ignore them.

When king \( i \) receives a message asking him to join the kingdom of king \( j \), he does so if \( \langle \phi(i), k(i) \rangle < \langle \phi(j), k(j) \rangle \) (lexicographically; namely, if either (a) \( \phi(i) < \phi(j) \) or (b) \( \phi(i) = \phi(j) \) and \( k(i) < k(j) \)).

The process of joining \( j \)'s kingdom is combined of two stages: first \( i \) sends a message to \( j \) along the same path which transferred \( j \)'s message to \( i \), telling him he is willing to join his kingdom; during this stage the directions of the edges in this path are reversed. In the second stage, if \( \phi(i) < \phi(j) \) then king \( j \) announces his new citizens that he is their new king, and if \( \phi(i) = \phi(j) \) then he increases his phase by 1 and sends an appropriate message towards all his citizens (new and old).

Six kinds of messages are used in this algorithm:

1. *Wake* this message, from some outside source, wakes a node and makes him start his algorithm. At most one such message can reach any node.

2. *Ask* \( \langle \phi(i), k(i) \rangle \) this message is sent by king \( i \) through an unused edge, in an attempt to increase his kingdom.
(3) \(\text{ACCEPT}(\text{phase}(j))\) : this message is sent by king \(j\) in return to an \textit{ASK} message from another king, telling him that he is willing to join his kingdom.

(4) \(\text{UPDATE}(\text{phase}(i), k(i))\) : this message is sent by king \(i\) (after receiving an \textit{ACCEPT} message from another king) updating his new (and in some cases also his old) citizens of his identity and phase.

(5) \(\text{YOUR_CITIZEN}\) : this message is returned by a citizen upon receiving an \textit{ASK} message originated by his own king.

(6) \(\text{LEADER}\) : this message is sent by the leader to all other nodes, announcing his leadership and terminating the algorithm.

We now give the formal description of the algorithm to be performed by a node \(i\) (as long as he is a king). \textit{unused}(i) denotes the set of all his unused edges, and initially contains all his \(n-1\) adjacent edges. \textit{receive}(m) means that if the queue of received messages is not empty, then \(m\) is the first message in it, else the processor waits until he receives a message \(m\).

The Algorithm for a king

begin

\(\text{receive}(m);\) \([m\ \text{will be either \textit{WAKE} or \textit{ASK}}]\)

\(\text{if } m = \text{ASK}(\text{phase}(j), k(j))\ \text{and} (\text{phase}(j), k(j)) > (0, \text{identity}(i))\)

\(\text{then } \text{state} := \text{CITIZEN};\) \([\text{else you remain a king}]\)

\(\text{while } (\text{unused}(i) \neq \emptyset \text{ and } \text{state} = \text{KING}) \text{ do}\)

\(\text{begin}\)

\(\text{choose } e \in \text{unused}(i);\)

\(\text{send an ASK message along } e;\)

\(\text{unused}(i) := \text{unused}(i) - \{e\};\)

\(\text{label: receive}(m).\)

\([m\ \text{will be one of the following}:\)

\(\text{LEADER, YOUR_CITIZEN, ACCEPT, ASK}]\)

\(\text{case } m \text{ of}\)

LEADER: stop; [you’ve just received a note from the (unique) leader]

YOUR_CITIZEN:
[do nothing and enter the while loop again]

ACCEPT(\text{phase}(j)):

\textbf{if} \ \text{phase}(i) > \text{phase}(j) \ \textbf{then}

\hspace{1cm} \text{send } \text{UPDATE}(\text{phase}(i),k(i)) \text{ along the edge that delivered this ACCEPT message}

\textbf{else} [i.e., \text{phase}(i) = \text{phase}(j)].

\hspace{1cm} \textbf{begin}

\hspace{2cm} \text{phase}(i) := \text{phase}(i) + 1;

\hspace{2cm} \text{send } \text{UPDATE}(\text{phase}(i),k(i)) \text{ to all your sons [new and old]}

\hspace{1cm} \textbf{end};

ASK(\text{phase}(j),k(j)):

\textbf{if} (\text{phase}(i),k(i)) > (\text{phase}(j),k(j)) \ \textbf{then} \text{ ignore this message and goto label}

\textbf{else}

\hspace{1cm} \textbf{begin}

\hspace{2cm} \text{send } \text{ACCEPT}(\text{phase}(i)) \text{ along the edge that delivered this message;}

\hspace{2cm} \text{[this edge leads from now on directly to your father, and towards your (new) king]}

\hspace{2cm} \text{state} := \text{CITIZEN}

\hspace{2cm} \text{[Sorry, you are no more a king!]}

\hspace{1cm} \textbf{end}

\textbf{end} [of the case statement]
end; [of the while loop]
[now state = CITIZEN or unused(i) = φ]
if state = CITIZEN
  then perform the procedure for a citizen
else send a message LEADER to all other nodes
  [Congratulations; you are the (only) leader!]
end.

The algorithm for a citizen (given in the full paper) imitates, in a sense, the algorithm for his king (with some exceptions; in particular, a citizen does not send messages along unused edges).

**Theorem:** The above algorithm finds a unique leader in $O(n \log n)$ messages.

In the analysis of the algorithm we charge messages to nodes in such a way that each node is charged with at most a certain constant number of messages per phase; we also prove that the number of phases is at most $\log n$. This proves the $O(n \log n)$ performance of our algorithm.

**Acknowledgement:** we would like to thank Doron Rotem for a discussion which led to this research.
REFERENCES


