TIGHT LOWER AND UPPER BOUNDS
FOR SOME DISTRIBUTED ALGORITHMS
FOR A COMPLETE NETWORK OF PROCESSORS

by

E. Korach, S. Moran and S. Zaks

Technical Report #423

June 1986
TIGHT LOWER AND UPPER BOUNDS
FOR SOME DISTRIBUTED ALGORITHMS
FOR A COMPLETE NETWORK OF PROCESSORS


Computer Science Department
Technion - Israel Institute of Technology
Haifa, Israel 32000
Tight Lower and Upper Bounds
for Some Distributed Algorithms
for a Complete Network of Processors

E. Korach, S. Moran and S. Zaks

Computer Science Department
Technion - Israel Institute of Technology
Haifa, Israel 32000

ABSTRACT
Distributed algorithms for complete asynchronous networks of processors (i.e., networks where each pair of processors is connected by a communication line) are discussed. The main results are an $O(n \log n)$ algorithm and an $O(n \log n)$ lower bound on the number of messages required by any algorithm in a given class. This class includes algorithms for problems like finding a leader or constructing a spanning tree (all previously known algorithms for those problems may require $O(n^2)$ messages when applied to complete networks). $O(n^2)$ bounds for other problems, like constructing a maximal matching or a Hamiltonian circuit are also given. In proving the lower bound we are counting the edges which carry messages during the executions of the algorithms (ignoring the actual number of messages carried by each edge). Interestingly, this number is shown to be of the same order of magnitude as the total number of messages needed by these algorithms. Moreover, the proofs of the lower bounds apply also for synchronous networks. In the upper bounds, the length of any message is at most $\log_2(4m \log_2 n)$ bits, where $m$ is the maximum identity of a node in the network. One implication of our results is that finding a spanning tree in a complete network is easier than finding a minimum weight spanning tree in such a network, which may require $O(n^2)$ messages.
1. INTRODUCTION

The model under investigation is a network of \( n \) processors with distinct identities \( \text{id}(1) \), \( \text{id}(2) \), \( \ldots \), \( \text{id}(n) \). No processor knows any other processor's identity. Each processor has some communication lines, connecting it to some others. The processor knows the lines connected to itself, 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 on each edge arrive, with no error, in a finite time, and are kept in order 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 it.

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 \cite{GHS} in \( O(n \log n + |E|) \) messages for a general graph. A leader in a network is found in \cite{G}, where \( n \) is known to every processor, in an expected number of messages which is \( O(n \log n) \) (independent of \( |E| \)); the worst case is not analyzed (but is said to be \( O(n |E|) \)).

An \( \Omega(n \log n) \) lower bound and an \( O(n \log n) \) upper bound for the message complexity of the problem of distributively finding a leader in a circular network of processors are known; see \cite{B, PKR} for the lower bound and \cite{DKR, ES, P} for the upper bound.

We address two classes of algorithms for complete graphs: the first must use edges of a spanning subgraph in every possible execution, and the second must use edges of a maximum matching in every possible execution. The problems of choosing a leader, finding a maximum and constructing a spanning tree clearly require algorithms that belong to the first class, while finding a complete matching or constructing a
Hamiltonian cycle clearly require algorithms that belong to the second class.

We prove a (worst case) lower bound of $\Omega(n \log n)$ for the number of edges (hence messages) used by any algorithm in the first class and a lower bound of $\Omega(n^2)$ edges for the second class; these proofs apply also for synchronous networks. An algorithm of $O(n^2)$ messages can easily be designed for the second class.

Next we present an algorithm that attains the 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 (minimum) identity and constructing a spanning tree. The correctness of the algorithm is proved and its complexity is analyzed. This algorithm together with the lower bound of $\Omega(n^2)$ for finding a minimum weight spanning tree presented in [KMZ2] show that in complete networks it is easier to find a spanning tree than to find a minimum weight spanning tree.

Our algorithms 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 the algorithms discussed in most other works in this area; in fact, we show that $|E| - 1$ messages are required for similar algorithms on a certain class of "almost complete" graphs, in which the ratio between the number of edges and $\binom{n}{2}$ tends to one as $n$ tends to infinity. This implies that almost $|E|$ messages may be required by any such algorithm, even when the underlying graph is known to be extremely dense (but not necessarily complete).

2. LOWER BOUNDS

In this section we study lower bounds for global algorithms and for matching algorithms (to be defined later). The proofs apply for both asynchronous and synchronous algorithms. For the synchronous case, the assumption that nodes can be awakened spontaneously at any time is crucial (e.g., if all nodes that spontaneously start the algorithm are awakened within a constant interval of time, an $O(n)$ upper bound for
2.1. Definitions and Axioms

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. With each execution we can associate a sequence \( \text{SEND} = \langle \text{send}_1, \text{send}_2, \ldots, \text{send}_k \rangle \) 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). In the case that two or more messages are sent at the same time, order them randomly (thus, in such case many sequences \( \text{SEND} \) may correspond to the same execution). Each event \( \text{send}_i \) we identify with the triple \( (v(\text{send}_i), e(\text{send}_i), m_i) \), where \( v(\text{send}_i) \) is the node sending the message \( m_i \) and \( e(\text{send}_i) \) is the edge used by it. We assume that \( \text{send}_1 \) occurred at time \( 0 \), and \( \text{send}_i \) at time \( \tau_i \geq 0 \).

Let \( \text{SEND}(t) \) be the prefix of length \( t \) of the sequence \( \text{SEND} \), namely \( \text{SEND}(t) = \langle \text{send}_1, \ldots, \text{send}_t \rangle \) (\( \text{SEND}(0) \) is the empty sequence). If \( t < t' \) then we say that \( \text{SEND}(t') \) is an extension of \( \text{SEND}(t) \), and we denote \( \text{SEND}(t) < \text{SEND}(t') \). \( \text{SEND} \) is called a completion of \( \text{SEND}(t) \). Note that a completion of a sequence is not necessarily unique.

Let \( \text{NEW} = \text{NEW}(\text{SEND}) \) be the subsequence \( \langle \text{new}_1, \text{new}_2, \ldots, \text{new}_r \rangle \) of the sequence \( \text{SEND} \) that consists of all the events in \( \text{SEND} \) that use previously unused edges. (An edge is used if a message has been already sent along it from either side.) This means that the event \( \text{send}_i = (v(\text{send}_i), e(\text{send}_i), m_i) \) belongs to \( \text{NEW} \) if and only if \( e(\text{send}_i) \neq e(\text{send}_j) \) for all \( j < i \). \( \text{NEW}(t) \) denotes the prefix of size \( t \) of the sequence \( \text{NEW} \).

Define the graph \( G(\text{NEW}(t)) = (V,E(\text{NEW}(t))) \), where \( E(\text{NEW}(t)) \) is the set of edges used in \( \text{NEW}(t) \), and call it the graph induced by the sequence \( \text{NEW}(t) \). If for every execution of the algorithm \( A \) the corresponding graph \( G(\text{NEW}) \) is connected then we term this algorithm global. Note that all the graphs \( G(\text{NEW}) \) above have a fixed set \( V \) of vertices (some of which may be isolated).
Let $NEW_i(t_i)$ be a prefix of a sequence $NEW_i$ in which the last event occurred at time $\tau_i$ for $i = 1, 2$, where $\tau_1 \leq \tau_2$. The synchronous merge $NEW_1(t_1) \circ NEW_2(t_2)$ of $NEW_1(t_1)$ and $NEW_2(t_2)$ is a sequence $s_1, \ldots, s_{i+2}$ obtained by associating with each event in $NEW_1(t_1)$ that occurred in time $\tau$ the time $\tau + (\tau_2 - \tau_1)$, and then merging them in any order that is consistent with the new timing of the events. This merge modifies the timing of the messages so as to terminate their execution at the same time. Note that a synchronous merge of two sequences that correspond to legal executions of the algorithm does not necessarily correspond to a legal execution, as the two original executions may conflict with each other; however, in certain cases, as in axiom 2 below, the synchronous merge yields a legal sequence.

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 $\sigma = NEW(t)$ corresponding to possible executions of $A$.

For the model used in this paper the following facts - defined below as axioms - hold for every algorithm $A$ and every graph $G$:

**axiom 1**: the empty sequence is in $EX(A,G)$.

**axiom 2**: if two sequences $\sigma_1$ and $\sigma_2$, which do not interfere with each other, are in $EX(A,G)$, then so is also their synchronous merge $\sigma_1 \circ \sigma_2$. ($\sigma_1$ and $\sigma_2$ do not interfere with each other if no two edges $e_1$ and $e_2$ that occur in $\sigma_1$ and $\sigma_2$ respectively have a common end point; this means that the corresponding partial executions of $A$ do not affect each other and hence any of their synchronous merges corresponds to a legal execution of $A$).

**axiom 3**: if $\sigma$ is a sequence in $EX(A,G)$ with a last element $(v,e,m)$, where $e$ was previously unused, and if $e'$ is an unused edge adjacent to $v$, then the sequence obtained from $\sigma$ by replacing $e$ by $e'$ is also in $EX(A,G)$. (This reflects the fact that a node cannot distinguish between its unused edges.)

Note that the above three axioms do not imply that $EX(A,G)$ contains any non-empty sequence. However, if the algorithm $A$ is global then the following fact holds as

\[\text{These axioms reflect only those properties of distributed algorithms which are needed here.}\]
well.

axiom 4 if $a$ is in $EX(A,G)$, such that $G(a)$ is not connected, and $C$ is a non-empty sub-set of $V$ containing all the non-isolated nodes in $G(a)$, then there is an extension $a'$ of $a$ in which the first event $(v,e,m)$ in $a'$ but not in $a$ satisfies $v \in C$. (This reflects the facts that some unused edge will eventually carry a message and that isolated nodes in $G(a)$ may remain asleep until some message from already awakened nodes will reach them).

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

2.2. Lower Bound for Global Algorithms

The following lemma is needed in the sequel:

Lemma 1: Let $A$ be a global algorithm acting on a complete graph $G=(V,E)$, and let $U \subseteq V$. Then there exists a sequence of messages $a$ in $EX(A,G)$ such that $G(a)$ has one connected component whose set of vertices is $U$ and the vertices in $V-U$ are isolated.

Proof: A desired sequence $a$ can be constructed in the following way. Start with the empty sequence (using axiom 1). If $|U| = 1$ then we are done; otherwise, add a message along a new edge that starts in a vertex in $U$ (axiom 4) and that does not leave $U$ (axiom 3 and the completeness of $G$). This is repeated until a graph having the desired properties is constructed.

Theorem 1: Let $A$ be a global algorithm acting on a complete graph $G$ with $n$ nodes. Then the edge complexity $e(A)$ of $A$ is $\Omega(n \log n)$.

Proof: For a subset $U$ of $V$ we define $e(U)$ to be the maximal length of a sequence $a$ in $EX(A,G)$ which induces a graph that has a connected component whose set of vertices is $U$ and isolated vertices otherwise (such a sequence exists by lemma 1).
Define \( e(k) \), \( 1 \leq k \leq n \), by
\[
e(k) = \min \{ e(U) \mid U \subseteq V, |U| = k \}^2
\]
Note that \( e(n) \) is a lower bound on the edge complexity of the algorithm \( A \).

The theorem will follow from the inequality
\[
e(2k+1) \geq 2e(k) + k + 1 \quad (k < \frac{n}{2})
\]

Let \( U \) be a disjoint union of \( U_1, U_2 \) and \{v\}, such that \( |U_1| = |U_2| = k \), and \( e(U) = e(2k+1) \). We denote \( C = U_1 \cup U_2 \).

Let \( \sigma_1 \) and \( \sigma_2 \) be sequences in \( EX(A,G) \) of lengths \( e(U_1), e(U_2) \) inducing subgraphs \( G_1, G_2 \) that have one connected component with vertex set \( U_1, U_2 \) (and all other vertices are isolated), respectively (these sequences exist by Lemma 1). \( \sigma_1 \) and \( \sigma_2 \) do not interfere with each other, and therefore - by axiom 2 - their synchronous merge \( \sigma = \sigma_1 \sigma_2 \) is also in \( EX(A,G) \). The set \( C \) satisfies the assumptions of axiom 4. Note that each node in \( C \) has at least \( k \) adjacent unused edges connecting it to other nodes in \( C \).

By axiom 4 there is an extension of \( \sigma \) by a message \((v,e)\), where \( v \in C \). By axiom 3 we may choose the edge \( e \) to connect two vertices in \( C \). This process can be repeated until at least one vertex in \( C \) saturates all its edges to other vertices in \( C \). This requires at least \( k \) messages along previously unused edges. One more application of axiom 4 and axiom 3 results in a message from some node in \( C \) to the vertex \( v \). The resulting sequence \( \sigma' \) induces a graph that contains one connected component on the set of vertices \( U \) and isolated vertices otherwise. Thus we have
\[
e(2k+1) = e(U) \geq e(U_1) + e(U_2) + k + 1 \geq 2e(k) + k + 1.
\]

The above inequality implies that for \( n = 2^i - 1 \) and the initial condition \( e(1) = 0 \) we have
\[
e(n) \geq \frac{n+1}{2} \log \left( \frac{n+1}{2} \right).
\]
Since it is obvious that \( e(m) \approx e(n) \) for \( m > n \), this implies the theorem. \( \square \)

From this theorem it follows that

\[\text{In general, one expects } e(k) = e(U) \text{ for any subset } U \text{ of } k \text{ vertices. However, one may easily construct algorithms for which } e(U_1) \neq e(U_2) \text{ for two distinct subsets } U_1 \text{ and } U_2 \text{ of equal cardinality.}\]
Theorem 2: Let $A$ be a global algorithm acting on a complete graph $G$ with $n$ nodes. Then the message complexity $m(A)$ of $A$ is $\mathcal{O}(n \log n)$.

Note 1. The lower bounds in Theorems 1 and 2 hold even in the case when every node knows the identities of all other nodes (but cannot tell which edge leads to which node).

Note 2: In the example constructed in the proof of Theorem 1 the number of processors which initialize the algorithm is $O(n)$ (it equals $\frac{n+1}{2}$ for $n = 2^i - 1$). In fact, $O(n)$ initiators are essential for any such example, since in the next section we prove an upper bound of $O(n \log k)$ messages for global algorithms, where $k$ is the number of the initiators of the algorithm.

Note 3: The $\Omega(n \log n)$ bound for the synchronous case is a simple extension of the one for the asynchronous case, presented in [KMZ1], and was independently pointed out in [AG2].

2.3. Lower Bounds for Matching-Type Algorithms

The above theorems imply that algorithms for tasks like constructing a spanning tree, finding the maximum identity, finding a leader, constructing a Hamiltonian path or constructing a maximum matching\(^3\) have a lower bound of $\Omega(n \log n)$ edges (and messages); however, for the last two cases we show even a stronger result. Let a matching-type algorithm be an algorithm that is guaranteed to cover a maximum matching (that is, to induce a graph which contains a matching of size $\left\lfloor \frac{n}{2} \right\rfloor$, where $|z|$ is the largest integer not larger than $z$).

Theorem 3: Let $A$ be a matching-type algorithm acting on a complete graph $G$ with $n$ nodes. Then the edge complexity $e(A)$ of $A$ is $\Omega(n^2)$.

Proof: Let $A$ be a matching-type algorithm. We construct a sequence in $EX(A,G)$ of length $O(n^2)$. Arbitrarily number the vertices from 1 to $n$. We construct the sequence $\sigma$ in the following manner:

Let $\sigma_0$ be the empty sequence. For $i \geq 0$ if $G(\sigma_i)$ does not contain a maximum matching, then $\sigma_{i+1}$ is an extension of $\sigma_i$ by a message $(v,e)$ where $e = (v,j)$ is chosen with

\(^3\) It is not hard to see that an algorithm that is guaranteed to construct a maximum matching must be global for complete graphs of $n$ vertices for even $n$, and to induce connected graphs of at least $n-1$ vertices for odd $n$. 
smallest possible \( j \) (we use here axiom 1, axiom 3 and an appropriate variant of axiom 4 for matching-type algorithms).

By the assumption that \( A \) is a matching-type algorithm, a sequence \( \sigma \) in \( EX(A,G) \) that does contain a maximum matching is eventually constructed. Let this matching be \( \{(u_i,v_i) \mid 1 \leq u_i < v_i \leq n \text{ and } u_i < u_{i+1}\} \).

Let \( n_i \) be the number of messages in \( \sigma \) which use an edge that connects \( u_i \) or \( v_i \) to some \( j < u_i \). By the construction of \( \sigma \), \( n_i \geq u_i - 1 \geq i - 1 \). Thus the length of \( \sigma \) is greater than

\[
0 + 1 + \cdots + \left( \frac{n_i}{2} \right) - 1 = \Omega(n^2).
\]

(Note that we did not count the edges \( (u_i,v_i) \) of the matching). This completes the proof. \[ \square \]

Note that the bound above shows that at least approximately \( \frac{n^2}{8} \) (about one half of the) edges carry messages in each execution. From this theorem it follows that

**Theorem 4:** Let \( A \) be a matching-type algorithm acting on a complete graph \( G \) with \( n \) nodes. Then the message complexity \( m(A) \) of \( A \) is \( \Omega(n^2) \).

Note that Theorems 3 and 4 are independent of the number of initiators, which is not the case for Theorems 1 and 2.

In [GHS] it was noted that global algorithms in general graphs require at least \( |E| \) messages when the number of vertices is unknown. We conclude this section by observing that even when the numbers of nodes and edges are known - and in fact the graph is almost complete and known up to isomorphism - then at least \( |E| - 1 \) messages may be requires in the worst case. To see this, consider a complete graph of \( n \) nodes to which a new vertex \( v \) is added on some unknown edge (the resulting graph has \( n+1 \) vertices and \( \binom{n}{2} + 1 \) edges). Apply the algorithm on such a graph with \( v \) asleep, and as long as there are unused edges, assume that \( v \) is on one of them. Thus \( |E| - 1 \) edges must be used in order to wake the vertex \( v \).
3. UPPER BOUNDS

3.1. General Discussion

We proved in the previous section a lower-bound of $\Omega(n^2)$ for the maximum matching problem. An algorithm of $O(n^2)$ messages for this problem can be easily designed (for example, let each node send messages to all its neighbors, and then form the matching by increasing order of identities, such that the node with smallest identity matches the one with second smallest identity, etc.).

We also proved in the previous section a lower bound of $\Omega(n \log n)$ for problems like finding a leader. We present now an algorithm of $O(n \log n)$ messages for this task. This algorithm can be used to design global algorithms of $O(n \log n)$ messages for other problems (like constructing a spanning tree).

It is interesting to note that for both classes of algorithms, the given upper bounds show that the lower bounds on the number of edges used (i.e. the edge complexities, given in the previous section) are also tight lower bounds on the message complexities of algorithms in these classes.

3.2. Informal Description of the Algorithm

We now present and discuss an $O(n \log n)$ 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 $i$ is a king (i.e. $state(i) = KING$), and - except for one - everyone will eventually become a citizen (a citizen will never become a king again). A node $i$ with $state(i) = KING$ is called king$_i$. The algorithm starts by a WAKE message, received by any non-empty subset of nodes.

During the algorithm, each king is the root of a directed tree which is its kingdom. All the other nodes of this tree are citizens of this kingdom, and each node knows its father and sons. Each node $i$ also stores the identity king$(i)$ and the phase $p(i)$ of its king, which are updated during the execution of the algorithm. $status(i) = (p(i), k(i))$ is called the status of node $i$. We say that $status(i) < status(j)$ if either (a) $p(i) < p(j)$
or (b) \( p(i) = p(j) \) and \( k(i) < k(j) \). Before the algorithm starts, \( k(i) = \text{identity}(i) \) and \( p(i) = -1 \) for each node \( i \). The following variables are also used: \( \text{unused}(i) \) denotes the set of all unused edges of node \( i \), \( \text{father}(i) \) denotes the edge connecting \( i \) to its father, and \( \text{sons}(i) \) denotes the set of edges connecting \( i \) to its sons. When no confusion occurs, we shall denote by \( \text{father}(i) \) and \( \text{son}(i) \) the nodes at the other end of the corresponding edges.

A king is trying to increase its kingdom by sending messages towards other kings (possibly through their citizens), asking them to join, together with their kingdoms, its kingdom. A citizen, upon receiving a message originated by a king, delays it, ignores it, or transfers it to (or from) its king.

When king \( i \) receives a message asking it to join the kingdom of king \( j \), it does so, if \( \text{status}(i) < \text{status}(j) \). Otherwise, the message is simply ignored, and the sending king will eventually become a citizen. The process of joining \( j \)'s kingdom is combined of two stages: first king \( i \) sends a message to king \( j \) along the same path which transferred \( j \)'s message to \( i \), telling it that it is willing to join its kingdom; during this stage the directions of the edges in this path are reversed. In the second stage, if \( p(i) < p(j) \) then king \( j \) announces to its new citizens that it is their new king, and if \( p(i) = p(j) \) then it first increases its phase by 1 and then sends an appropriate updating message towards all its citizens (new and old).

3.3. The Messages used by the Algorithm

Six kinds of messages are used in this algorithm:

1. **Wake**: this message, from some outside source, wakes a node and makes it start its algorithm. Any node can get at most one such message.
2. **Ask** \((\text{status}(i))\) this message is sent by king \( i \), having a status \( \text{status}(i) \), through an unused edge in an attempt to increase its kingdom, and might be transferred onwards by citizens. \( \text{status}(i) \) is considered as the status of the \( \text{Ask} \) message.
3. **Accept** \((p(i))\) this message is sent by king \( i \), in phase \( p(i) \), in return to an \( \text{Ask} \) message from another king, telling it that it is willing to join its kingdom. (This message might also be transferred onwards by citizens.)
(4) \( UPDATE(status(i)) \): this message is sent by a king \( j \), having a status \( status(i) \) (after receiving an ACCEPT message originated by another king) updating its new (and in some cases also its old) citizens of its identity and phase.

(5) \( YOUR\_CITIZEN \): this message is returned by a citizen as an answer to an \( ASK \) message originated by its own king.

(6) \( LEADER \): this message is broadcasted by the leader to all other nodes announcing its leadership and terminating the algorithm.

3.4. The Algorithm for a King

Before the algorithm starts every node \( i \) has the following initial values. For convenience, we denote any local variable \( x \) at node \( i \) as \( x(i) \).

\[
\begin{align*}
state(i) &= KING, \\
status(i) &= (p(i), e(i)) = (-1, identity(i)), \\
enused(i) &= \text{the set of all its } n-1 \text{ adjacent edges.}
\end{align*}
\]

\[sons(i) = \emptyset.\]

We now give the formal description of the algorithms to be performed by node \( i \) as long as it is a king.

Node \( i \) reacts to a message \( m \) if receives along an edge \( e \) according to \( m \)'s type, as follows:

(K1) \( m = WAKE \): Node \( i \) increases its phase to zero and performs the procedure \( \text{check} \) (given below).

(K2) \( m = ASK(status(j)) \): If \( status(j) \leq status(i) \) then node \( i \) ignores this message.

Otherwise, it sends an \( ACCEPT(\phi(i)) \) message along \( e \), and changes its state to \( CITIZEN \), with \( father(i) = e \).

(K3) \( m = YOUR\_CITIZEN \): Node \( i \) performs the procedure \( \text{check} \).

(K4) \( m = ACCEPT(p(j)) \): Node \( i \) does the following:

(1) Adds \( e \) to \( sons(i) \).

(2) If \( p(j) < p(i) \) it sends an \( UPDATE(status(i)) \) message along \( e \). Otherwise (i.e.,

\[\text{Actually, it can be shown that } status(j) = status(i) \text{ may not happen.}\]
\( p(j) = p(i) \) it increases its phase by one and sends an \( UPDATE(status(i)) \) message along all the edges in \( sons(i) \).

(3) It performs the procedure check.

Procedure check:

- If \( unused(i) \neq \emptyset \) then delete an edge \( e' \) from \( unused(i) \) and send an \( ASK(p(i),k(i')) \) message along \( e' \).
- Else (you are the leader) send a \( LEADER \) message to all your sons and terminate your algorithm.

3.5. The Algorithm for a Citizen

The algorithm for a citizen is basically simple, since the only task of a citizen is passing messages to, or from, its king. However, doing it in the straightforward way might use \( O(n^2) \) messages. We incorporate some control mechanism into the algorithm, which enables us to reduce the message complexity to \( O(n \log n) \).

A node \( i \) in the \( CITIZEN \) state has a unique father edge leaving it (which may be changed during the algorithm) and zero or more son edges. It may be in one out of two substates: \textit{regular} and \textit{waiting}. It enters the \textit{waiting} substate when it becomes a citizen, or upon receiving an \( ASK \) message with status higher than its own status. It returns to the \textit{regular} substate upon getting a status which is higher or equal to the status of that \( ASK \) message. While in the \textit{waiting} substate, \( i \) remembers the status \( (p(j),k(j)) \) of the last \( ASK \) message it received, and also the edge \( e' \) along which it received this message. Also, in this substate it receives messages only from its father, and delays the reception of other messages (from other edges) until it enters the \textit{regular} substate (if at all).

The algorithm for a citizen follows:

Initialization:

\[ substate(i) = \text{waiting}, \]

Node \( i \) reacts to a message \( m \) it receives along an edge \( e \) according to \( m \)'s type and its

---

*The case \( p(j) > p(i) \) is not possible; this follows from (32), Lemma 5 and Corollary 2.*
substrate, as follows.

regular substate:

(CR1) \( m = \text{ASK}(\text{status}(j)) \): \( i \) does the following:

1) If \( k(i) = k(j) \) and \( e \) is not a son edge, it returns a \text{YOUR\_CITIZEN} message along \( e \)

2) If \( \text{status}(j) > \text{status}(i) \) and \( k(i) \neq k(j) \), it sends \( m \) to its father and enters the waiting substate.

(CR2) \( m = \text{UPDATE}(\text{status}(j)) \): It updates its status to \text{status}(j), and forwards \( m \) to all its sons.

(CR3) \( m = \text{ACCEPT}(p(j)) \): It makes \( e \) its son edge, and forwards an \text{UPDATE}(\text{status}(i)) message along \( e \).

(CR4) \( m = \text{LEADER} \): \( i \) sends this message to all its sons and terminates the algorithm.

(CR5) \( m = \text{YOUR\_CITIZEN} \) or \( m = \text{WAKE} \): \( i \) ignores this message.

waiting substate:

(Recall that: (1) \((p(j), k(j))\) is the status of the last ASK message received by \( i \), (2) \( e' \) is the edge along which this ASK message was received, and (3) in this state \( i \) receives messages only from its father):

(CW1) \( m = \text{UPDATE}(p(k), k(k)) \): \( i \) does the following:

1) updates its status to \((p(k), k(k))\) and forwards \( m \) to all its sons.

2) If \( k(k) = k(j) \) and \( e' \) is not a tree edge, it sends a \text{YOUR\_CITIZEN} message along \( e' \) and returns to the regular substate.

3) If \((p(k), k(k)) \geq (p(j), k(j))\) it returns to the regular substate

(CW2) \( m = \text{ACCEPT}(p(k)) \): Make \( e \) (which is your father) your son, make \( e' \) your father and forward \( m \) along \( e' \)

(CW3) \( m = \text{LEADER} \): \( i \) sends this message to all its sons and terminates the algorithm.

It can be shown that in this substate a node cannot receive a \text{YOUR\_CITIZEN} message (this fact, however, is not needed for the proof, as it can be assumed that messages of this type are ignored). In addition, no ASK message can be received by a node.
in this substate, as will be shown in Corollary 3.

3.6. Basic Properties of the Algorithm

The basic idea of the algorithm is building a spanning tree of the graph by repeatedly combining small trees to bigger ones. These trees will be viewed as directed trees, each such tree considered as a kingdom, with the root being its unique king. An edge \( e \) which is \( \text{father}(i) \) for some node \( i \) is called a tree edge, (and is called a non-tree edge otherwise), and is considered to be directed into \( i \). The only way for a non-tree edge to become a tree edge, or for a tree edge to reverse its direction, is as a result of an ACCEPT message sent along it: The sender makes this edge its father (by executing (K2) or (CW2)), and the receiver makes this edge its son (by executing (K4), (CR3) or (CW2)).

**Observation 1:** Let \( e \) be an edge connecting \( i \) and \( j \), then \( e \) is \( j \)'s son only if it is \( i \)'s father (the converse is not necessarily true).

**Lemma 2** An \( \text{ASK} \) message is originated by a king. The first edge, \( e \), that carries it is a non-tree edge, and the other edges are tree edges traversed against their directions.

**Proof:** An \( \text{ASK}(p(i),k(i)) \) message is originated by king \( j \) in the procedure check, and is forwarded from sons to fathers by (CR1).

Let \( \text{SEND} = <send_1,send_2, > \), where \( send_j = (v_j,e_j,m_j) \), be a sequence corresponding to an execution of the algorithm, as defined in Section 2.1, and let \( m_i = \text{ASK}(p(i),k(i)) \). Then, since every node that receives an \( \text{ASK} \) message forwards at most one such message (see (CR1)), the edges that carry this message form an (undirected) path. We shall denote this path by \( \pi(i,t) \). When no confusion occurs, we simply denote it by \( \pi(i) \).

**Lemma 3:** At most one \( \text{ASK} \) message is sent by any citizen during the algorithm at any given phase.

**Proof:** An \( \text{ASK} \) message is sent by a citizen only in (2) of (CR1). After sending an \( \text{ASK}(p,k) \) message, a citizen enters the waiting substate. While in this substate, it cannot send another \( \text{ASK} \) message, and it can exit this substate only upon updating its
status to \((p',k')\), where \(p' \geq p\). Since a phase of a node never decreases, it will never get a lower phase, and therefore, by (CR1), it will never forward any other \(ASK\) message at phase \(p\).

It follows easily from the above lemma, the fact that a citizen never becomes a king and the fact that a king never forwards an \(ASK\) message, that

**Corollary 1:** If an \(ASK(p(i),k(i))\) message is received by some king \(j\), then \(\pi(i)\) is a simple path.

**Lemma 4:** When a node \(v\) sends an \(UPDATE(p,k)\) message to node \(u\) along edge \(e\), \(e\) is \(u\)'s father and \(status(v) = (p,k)\).

**Proof:** An \(UPDATE\) message is sent only to sons ((K4), (CR2), (CR3) and (CW1)), and by Observation 1 the receiver must receive it from its father.

**Lemma 5:** Suppose that an \(ACCEPT(p(j))\) message is sent by king \(j\), with \((p(j),k(j)) = (p',k')\), in response to an \(ASK(p(i),k(i))\) message. Let \(u \in \pi(i), u \neq i,j\). Then, after \(u\) forwarded this \(ASK\) message, the following holds:

1. \(u\)'s status cannot exceed \((p',k')\) before it receives some \(ACCEPT\) message, and
2. \(u\) will receive the \(ACCEPT(p(j))\) message from its father.

**Proof:** (1): For contradiction, assume that \(u \neq i\) is the node closest to \(j\) (possibly \(j\) itself) in \(\pi(i)\) whose status becomes \((p'',k'') > (p',k')\) before it received any \(ACCEPT\) message after it forwarded the \(ASK(p(i),k(i))\) message. This could happen only if \(u\) received an \(UPDATE(p'',k'')\) message from its father, say \(v\). Since only \(ACCEPT\) messages from sons to fathers may change 'fathers, \(v\) was \(u\)'s father also when \(u\) received and forwarded the \(ASK(p(i),k(i))\) message. Hence \(v\) is in \(\pi(i)\), between \(u\) and \(j\), and thus closer than \(u\) to \(j\). Also by Lemma 4, \(v\)'s status was raised to \((p'',k'')\) before \(u\)'s status was. By the assumption on \(u\), \(v\)'s status became \((p'',k'')\) after it had sent the \(ACCEPT(p(j))\) message. This could happen only by receiving an \(UPDATE\) message from its father. But while sending the \(ACCEPT\) message, \(v\) made \(u\) its father. This means that \(u\)'s status was raised to \((p'',k'')\) before \(v\)'s status was - a contradiction.
(2): $u$ entered the waiting substate upon receiving the $ASK(p(i),k(i))$ message. By
(1), $u$ did not exit the waiting substate unless it received an $ACCEPT$ message. If $u$ did
not receive the $ACCEPT(p(j))$ message then there exists a node in $\pi(i)$ which did not
receive this message, but its father did receive it, which, by an argument similar to the
above, yields a contradiction.

By the above Lemma, and by (CW2), we get:

**Corollary 2:** An $ACCEPT$ message that is sent along an edge $e$ was originated in
response to the last $ASK$ message sent on $e$, and is sent in direction which is opposite
to that of the $ASK$ message.

**Corollary 3:** A citizen in the waiting substate cannot receive an $ASK$ message.

**Proof:** For contradiction, let $u$ be the first node to receive an $ASK(p',k')$ message while
it is in the waiting substate. Being in this substate, $u$ must have received this message
from its father, say $v$, along an edge $\ell$. This means that either (1) $v$ is a king and the
$ASK(p',k')$ message was sent by procedure check, and hence $e \in unused(v)$, or (2) $u$
is a citizen which is $v$'s father and the $ASK(p',k')$ message was sent by (CR1). The former
case is impossible since $e$ is a tree edge, and therefore cannot be in $unused(v)$ for a
king $v$ ($e$ became a tree edge only after an $ACCEPT$ message, and hence an $ASK$
message, was sent on it).

In the second case, let $ASK(p(i),k(i))$ be the message that caused $u$ to enter the
waiting substate (by executing (CR1)), and let $\pi(i)$ be the corresponding path. Then by
Corollary 2, $v$ is also in $\pi(i)$. Since $u$ is $v$'s father, $v$ must have had at this time sent
an $ACCEPT$ message in response to the $ASK(p(i),k(i))$ message. Since $v$ is $u$'s father,
$u$ have not yet received this $ACCEPT$ message. But this means that $u$ could not yet send an $UPDATE$ message to $v$, and hence that $v$ was in the waiting substate when it
had received (and sent) the $ASK(p',k')$ message, contradicting the assumption on $u$.

**Lemma 6:** The number of times $ACCEPT$ messages are sent in any execution of the
algorithm is finite.

**Proof:** This follows from Corollary 2, Lemma 3, and the fact that an $ACCEPT$ message is
originated by a king that becomes a citizen, and never becomes a king again.
With each sequence $SEND(t)$ that corresponds to a partial execution of the algorithm, as defined in the previous section, we associate a directed graph $F(SEND(t)) = F(t)$ that contains all the edges that carried an ACCEPT message. The direction of each edge is opposite to the direction of the last ACCEPT message it carried.

**Theorem 5:**

1. $F(t)$ is a directed forest.
2. If $F(t) \neq F(t-1)$, then $F(t)$ is obtained from $F(t-1)$ by reversing or adding one edge, $e$, that is adjacent to a root of one of the trees in $F(t-1)$.
3. Assume that $send_t$ is an $ACCEPT(p(j))$ message, originated by node $j$ and sent on a non-tree edge, $e$, from node $u$ to node $v$. Then in $F(t-1)$, $u$ and $v$ belong to different trees and $status(v) > status(j) \geq status(u)$.
4. Every king is a root and it has the maximal status in its tree.
5. Let $T$ be a tree in $F(t)$ that does not contain a king, and let $u$ be the root of $T$. Then for some $t' < t$, $send_t$ is an $ACCEPT$ message that was sent to $u$, but not yet forwarded by it, and the originator of this $ACCEPT$ message (i.e., the node that sent it by executing $(K2)$) has the maximal status in $T$.

**Proof:** By induction on $t$. For $t = 1$ the theorem is true, as $F(0)$ consists of isolated nodes, which are all kings, and $F(1) = F(0)$ since $send_1$ is an ASK message. Assume the theorem holds for $t' < t$, and prove it for $t$.

2. If $F(t) = F(t-1)$ then (2) trivially holds, so assume that $F(t) \neq F(t-1)$, which means that $send_t$ is an $ACCEPT(p(j))$ message, sent by a node $u$. It remains to prove that $u$ was a root in $F(t-1)$. If $u = j$ - the originator of this message - then the claim holds by the induction hypothesis on (4). Otherwise, by Lemma 5, before sending that message, $u$ received it from its father, $u'$, which, by induction, was then a root. By the definition of $F(t)$, $u$ became a root at the time $u'$ sent this message.

3. Assume that $send_t$ is an $ACCEPT(p(j))$ message that was originated by $j$ when its status was $(p', k')$. This message is a response to an $ASK(p', k')$ message originated by
some node $i$, with $(p', k') > (p, k')$. By Corollary 2, $e$ must belong to $\pi(i)$. By the fact that $e$ is a non-tree edge, $i = v$. By Lemma 5, $\text{status}(u)$ was not larger than $(p', k')$ in $F(t-1)$, hence also in $F(t)$ (recall that $\text{send}_t$ is an ACCEPT message). Since $(p', k') \leq \text{status}(v)$ (status never decreases), we have that

$$\text{status}(u) \leq (p', k') < (p', k') \leq \text{status}(v).$$

By the induction hypothesis on (5), no node in the tree containing $u$ in $F(t-1)$ has a status larger than $(p', k')$, hence $u$ is not in this tree.

(1) If $F(t) = F(t-1)$ then (1) clearly holds. Otherwise, by (2) and the definition of $F(t)$, $\text{send}_t$ is an ACCEPT message sent along an edge $e$ adjacent to a root. If $e$ is a tree edge in $F(t-1)$, then, by (2), its direction was reversed and $F(t)$ is clearly a forest. Otherwise, by (3), $e$ connects two disjoint trees in $F(t-1)$, and by (2) the graph obtained by this connection is a directed tree.

(4) If a node $i$ is a king, then $\text{SEND}(t)$ does not contain an ACCEPT message sent by $i$; hence in $F(t)$ there is no edge entering $i$, and this - together with (2) - implies that $i$ is a root of a tree in $F(t)$.

Suppose that the maximal status $(p, k)$ of a node in a tree $T$ in $F(t)$ is greater than the status of the king $\text{king}(T)$ in $T$. Let $u$ be a node closest to the root of $T$ with status $(p, k)$.

If $u$ was in the same tree as the node $\text{king}(T)$ in $F(t-1)$ then, by (4), $\text{status}(\text{king}(T)) \geq \text{status}(u)$ in $F(t-1)$. If the status of $u$ was increased, then it must have been as a result of an UPDATE message from its father, that has already at least the same status (by Lemma 4), a contradiction.

If $u$ was in $F(t-1)$ in a tree $T'$ that is distinct from the tree $T''$ that contained $\text{king}(T)$, then the last message is clearly an ACCEPT message from a node $w$ in $T'$ to a node $v$ in $T''$. By (3) and (5) it follows that in $F(t)$ $\text{status}(u) > \text{status}(z)$ for every node $z$ in $T'$. In particular, $\text{status}(v) > \text{status}(u)$, a contradiction.

(5) Let $\hat{T}$ be the tree that contained $u$ in $F(t-1)$ (note that $V(\hat{T}) \subseteq V(T)$). If $\hat{T}$ contained a king, then that king had a maximal status in $\hat{T}$, and $\text{send}_t$ is an accept message sent
by this king to $u$, and the claim holds.

If $\hat{T}$ does not contain a king, then by induction, the root of $\hat{T}$, $\hat{u}$, satisfies the claim. If $\hat{u} = u$, then it suffices to show that the maximal identity of a node in $T$ equals the maximum identity of a node in $\hat{T}$ if $V(\hat{T}) = V(T)$ then this is trivial, since the only way for some node with a maximal identity in $\hat{T}$ to increase its phase is by executing (K4) as a king, and $\hat{T} = T$ contains no king. Otherwise, $send_{\hat{u}}$ is an ACCEPT message sent by a node $\hat{u}$ in $\hat{T}$ to a node $v$ in $\hat{T}$. Let $j$ be the originator of this ACCEPT message. Then, by the assumption that (5) and (3) hold for $t-1$, $j$ has the maximal status in $\hat{T}$, and $\text{status}(j) < \text{status}(v)$. This implies that the maximum identity of a node in $\hat{T}$ is smaller than the maximum identity of a node in $\hat{T}$, and the claim holds.

From Lemma 6 and the above theorem, we have

**Corollary 4:** There is a $t_0$ such that

(1) For all $t \geq t_0$, $F(t) = F(t_0)$, and

(2) Every root in $F(t_0)$ is a king.

**Proof:** Let $t_0$ be such that $send_{t_0}$ is the last ACCEPT message sent by the algorithm (such $t_0$ exists by Lemma 6). (1) follows by the fact that if $F(t) \neq F(t-1)$ then $send_{t}$ must be an ACCEPT message. (2) follows from the fact that if there exist a root in $F(t_0)$ that is not a king, then by (5) of Theorem 5 another ACCEPT message will be sent, a contradiction.

3.7. Correctness of the Algorithm

We prove in this section the correctness of the algorithm. The property which implies this correctness is given in the next theorem.

**Theorem 6:** In any execution of the algorithm, eventually there is exactly one node with status = KING
**Proof:** For contradiction, consider an execution for which the theorem is false. Let \( t_0 \) be as in Corollary 4. Then the following must hold.

\((**): F_{t_0}\) contains \( s > 1 \) nodes that remain kings forever. These nodes are the roots of disjoint trees \( T_1, \ldots, T_s \) such that \( \bigcup_{i=1}^s V(T_i) = V \).

Denote these nodes \( \text{king}_1, \ldots, \text{king}_s \). The proof proceeds by few lemmas.

**Lemma 7:** For \( i = 1, \ldots, s \), each \( \text{king}_i \) has its phase eventually equal to some constant \( \text{phase}_i \) forever.

**Proof:** It suffices to show that the number of phase-increases by kings is bounded. A king may increase its phase in \((K1)\) or in \((K4)\), after receiving a \( \text{WAKE} \) or an \( \text{ACCEPT} \) message, correspondingly. The number of \( \text{WAKE} \) messages is clearly bounded by \( n \).

By Lemma 6, the number of distinct \( \text{ACCEPT} \) messages is also bounded.

Let \( s_i \) denote the final status of \( \text{king}_i \) (such a status exists by the above lemma) and suppose that \( s_i < s_j \) for \( i < j \).

**Lemma 8:** Under the assumption \((**), \) eventually every node in the tree \( T_i \) will have its status equal to \( s_i \).

**Proof:** For contradiction, assume that some node \( j \) in the tree \( T_i \) has a different status \((p(j), k(j))\) forever. Assume that \( \tilde{j} \) is the node closest to the root having this property, let \( z \) be \( \tilde{j} \)'s father and \( e \) be the edge connecting \( z \) and \( \tilde{j} \). Then \( \text{status}(z) = s_i > (p(j), k(j)) \). \( z \) got this status by receiving an \( \text{UPDATE}(s_i) \) message. If at the time \( z \) received this message \( e \) was in \( \text{sons}(z) \), then \( z \) would have forwarded that \( \text{UPDATE} \) message towards \( \tilde{j} \) (CR2). Otherwise, \( z \) made \( e \) its son by receiving an \( \text{ACCEPT} \) message along it, and then sent an \( \text{UPDATE}(s_i) \) along it (CR3). In both cases, an \( \text{UPDATE}(s_i) \) message was eventually received by \( \tilde{j} \), a contradiction.

**Lemma 9:** Under the assumption \((**), \) \( \text{king}_s \) eventually sends an \( \text{ASK} \) message to a node which is eventually in \( T_j \) for \( j < s \).

**Proof:** Consider the last \( \text{ASK} \) message sent by \( \text{king}_s \) (there must be such a message, since at most \( n-1 \) \( \text{ASK} \) messages are sent by any king), and let \( u \) be the node that
received this message. For contradiction, assume that \( u \) is eventually in \( T_s \). If upon receiving this message \( \text{status}(u) = s_s \) then it will immediately return a \( \text{YOUR_CITIZEN} \) message to \( \text{king}_s \) (CR1). Otherwise, by Corollary 3, \( u \) must be in the regular substate, and it will forward that \( \text{ASK} \) message to its father and will enter the waiting substate (CR1). \( u \) can exit the waiting substate only by receiving an \( \text{UPDATE}(s_s) \) message (CW1). By Lemma 8, \( u \) will eventually receive this message, and will then send a \( \text{YOUR_CITIZEN} \) message to \( \text{king}_s \). In both cases, \( s \) will receive a \( \text{YOUR_CITIZEN} \) message. Since \( \text{unused}(s) \) is not empty (\( s \) did not use any edge connecting it to \( T_j \) for \( j < s \)), there are still some unused edges leaving \( s \). \( s \) will therefore initiate another \( \text{ASK} \) message, a contradiction.

**Lemma 10:** Eventually, no citizen is in the waiting substate.

**Proof:** For contradiction, let \( u \) be a citizen in \( T_j \) which is in the waiting substate forever, but \( u \)'s father (in \( T_j \)), say \( v \), is not. Assume that \( u \) entered this substate by receiving an \( \text{ASK}(p,i) \) message, which it forwarded to a node \( v' \), that was \( u \)'s father at that time.

After receiving this \( \text{ASK} \) message, \( u \) did not receive an \( \text{UPDATE}((\text{status}(l))) \) message with \( \text{status}(l) \geq (p,i) \), as otherwise, by (3) of (CW1), it would have returned to the regular substate.

Moreover, after receiving this \( \text{ASK} \) message, \( u \) also did not receive any \( \text{ACCEPT} \) message: by Corollary 2, the only \( \text{ACCEPT} \) message it could receive was in response to that \( \text{ASK}(p,i) \) message, and if such an \( \text{ACCEPT} \) message was sent, it would re-traverse the path \( \pi(i) \) and reach \( i \), the originator of the \( \text{ASK} \) message. Upon receiving this message, \( i \) would have initiated \( \text{UPDATE} \) messages. At the time when \( i \) initiated these messages, it was a root of the tree that contained also \( u \); This follows by (4) of Theorem 5, and the fact that when this \( \text{ACCEPT} \) message reached \( i \), all the edges between \( u \) and \( i \) in \( \pi(i) \) were tree edges (directed from \( i \) to \( u \)). Moreover, while delivering this \( \text{ACCEPT} \) message, all the citizens in \( \pi(i) \) were in the waiting substate, and hence could not receive any \( \text{ASK} \) (and hence \( \text{ACCEPT} \)) message before leaving this substate. This implies that the direction of the edges in \( \pi(i) \) could not be changed before the above
UPDATE message was received by u. This would imply that u left this waiting sub-state, a contradiction.

It follows from the above that \( v' = v \), and hence \( v' \) received this ASK message from u. \( v' \) did not forward this ASK message, in spite of the fact that it is in the regular sub-state. This could happen only if \( \text{status}(v') \geq (p, t) \). But eventually \( \text{status}(v') = \text{status}(u) \), and hence \( \text{status}(u) \) is also greater than \( (p, t) \), which means that it had to leave the waiting sub-state, a contradiction.

**Lemma 11:** Suppose king, eventually sends an ASK message to a node \( u \) which is in \( T_j \) for \( j < s \). Then \( u \) will eventually be in \( T_s \).

**Proof:** By Lemma 10, \( u \) will eventually exit the waiting sub-state. Since \( s_p \) is the maximum status in the network, this can happen only by having \( u \) receive an UPDATE(\( s_p \)) message (see (1) of (CW1)), which implies the lemma.

By Lemma 11 we get a contradiction to the assumption that \( s > 1 \), and this completes the proof of the theorem.

**Corollary 5:** The unique remaining king, say \( t \), eventually announces its leadership to all other nodes, and then the algorithm terminates.

**Proof:** By a proof similar to the one of Lemma 9, it can be shown that \( t \) will eventually exhaust all its unused edges by sending ASK messages and receiving YOUR_CITIZEN replies. This means that at this time all the nodes in the graph belong to the tree rooted at \( t \). When no unused edges remain, it will send the LEADER message to all its sons (by executing procedure check), which will then broadcast this message in the tree to all other nodes ((CR4) and (CW3)). Thus this message will reach each node, which will then terminate the algorithm.

**3.8. Complexity Analysis of the Algorithm**

We conclude by giving a complexity analysis of the algorithm as follows.

**Theorem 7:** If \( k \) nodes start the algorithm by a WAKE message, then the number of messages used by the algorithm is bounded by \( 5n \log_2 k + O(n) \).
We first need the following lemma:

**Lemma 12:** If $k$ nodes start the algorithm by *WAKE* messages and node $i$ is the leader, then when the algorithm stops we have

$$p(i) \leq \lfloor \log_k k \rfloor.$$

**Proof:** A king at phase $t$ increases its phase to $t+1$ only upon receiving an $\text{ACCEPT}(p(j))$ message, where $p(j) = t$. This $\text{ACCEPT}$ message was originated by king $j$ at phase $t$, who then became a citizen. Therefore, we have at most $\frac{k}{2}$ kings in phase 1, $\frac{k}{2^2}$ kings in phase 2, $\ldots$, $\frac{k}{2^l}$ kings in phase $l$, for every $1 \leq l \leq \lfloor \log_2 k \rfloor$, which implies the lemma.

**Proof of the theorem:** We give an upper bound for the number of messages of each kind:

1. **WAKE:** exactly $k$ messages.

2. **LEADER:** exactly $n-1$ messages.

3. **ASK:** at a given phase, a king with $m$ citizens can send at most $m+1$ such messages; therefore, all the kings in this phase sent together at most $n$ messages, hence the total number of ASK messages sent by kings is bounded by $n \log_2 k$. Every citizen transfers at most one ASK message per phase, hence the total number of such messages sent by all citizens is also bounded by $n \log_2 k$.

4. **YOUR_CITIZEN:** each such message is sent on an edge $e'$, in response to an ASK message that was previously sent on this edge ((1) of (CR1) and (2) of (CW1)). Therefore, the total number of such messages is bounded by the number of ASK messages originated by kings, which is $n \log_2 k$.

5. **ACCEPT:** the total number of such messages originated by kings is exactly $n-1$. By Corollary 2, the total number of such messages sent by all citizens is bounded by the total number of ASK messages sent by all citizens, hence it is also bounded by $n \log_2 k$. 

Technion - Computer Science Department - Technical Report CS0423 - 1986
(6) **UPDATE**: each citizen receives at most one such message per phase, hence the total number of such messages is also bounded by $n \log_2 k$.

To conclude, the total number of messages used by the algorithm does not exceed $5n \log_2 k + O(n)$.

A message of type *WAKE, YOUR_CITIZEN* and *LEADER* requires a constant number of bits each. A message of type *ASK, ACCEPT, UPDATE* requires 2 bits for specifying their type, $\log \log n$ bits for the phase, and $\log m$ bits for the identity (where $m$ is the largest identity). Therefore the maximal number of bits per message is bounded by $\log_2 [4m \log_2 n]$.

4. **SUMMARY**

In this paper we presented tight upper and lower bounds for the message complexity of certain problems regarding a distributed complete network of processors.

We addressed two classes of algorithms for complete graphs; one that must use edges of a spanning subgraph, and another that must use edges of a maximum matching. We showed a lower bound of $\Omega(n \log n)$ for the number of edges (hence messages) used by any algorithm in the first class and a lower bound of $\Omega(n^2)$ edges for the second class; the results apply also for synchronous networks.

Next we presented an algorithm that attains the bound of $O(n \log n)$ messages for the problem of choosing a leader in a complete graph. The correctness of the algorithm was proved in details, and its complexity was analyzed, showing that at most $5n \log k + O(n)$ messages are sent in any execution, where $k$ is the number of nodes that are spontaneously awakened. In [KMZ2] an $\Omega(n^2)$ lower bound is proved for the construction of a minimum-weight spanning tree. These results show, for the first time, that finding a minimum-weight spanning tree can be harder than finding any spanning tree in distributed networks.
The algorithm presented here is extended in [KM] to an algorithm for construction of a spanning tree in a general graph that requires at most $4n \log k + 2|E| + O(n)$ messages, where $k$ is the number of nodes that are spontaneously awakened.

In our model of the complete network it is assumed that all the edges adjacent to one node are indistinguishable. In [S] this situation is termed as having no sense of direction. Few other works belong here. In [M] and [MLW], a complete network is studied, where the processors have a full sense of direction; this means that they are arranged on a ring, and everyone knows where in the ring each of its adjacent edges is leading to (first node on the left, second node on the left, and so on). With this additional knowledge a leader can be found in $O(n)$ messages (in a recent work [ASZ] it is shown that it is sufficient to know $O(\log n)$ of the neighbors and still get a linear algorithm). Extensions of these results for various intermediate models are studied in [SUZ] and [HZ].

Our results about the complete network stimulated a lot of research in this area. Our $O(n \log n)$ upper bound for electing a leader was followed by independent works ([AG1], [H], [KKM] and [P]), where other $O(n \log n)$ algorithms are presented. The lower bounds, together with the ones in [KMZ2], are extended to various intermediate models in [SUZ] and [HZ].

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


