# Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of Processors

E. Korach
*IBM Israel Scientific Center, Technion City, Haifa, Israel 32000*

S. Moran
*Computer Science Department, Technion - Israel Institute of Technology, Haifa, Israel 32000*

S. Zaks†
*Laboratory for Computer Science, M.I.T., Cambridge, MA 02139*

## 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 result is $O(n\log n)$ lower and upper bounds on the number of messages required by any algorithm in a given class of distributed algorithms for such networks. This class includes algorithms for problems like finding a leader or constructing a spanning tree (as far as we know, all 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 actuall number of messages carried by each edge). Interestingly, this number is shown to be of the same order of magnitude of the total number of messages needed by these algorithms. 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.

† On leave from Computer Science Department, the Technion - Israel Institute of Technology; supported by NSF grant MCS-8302391.



## 1. INTRODUCTION

The model under investigation is a network of $n$ processors with distinct identities *identity*(1), *identity*(2), ... , *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 order in a list until processed (this list is not always treated as a queue). 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 [1,7] for the lower bound

and [2,4,5,8] for the upper bound. For a discussion of lower bounds on leader finding algorithms see [9].

We address two classes of algorithms for complete graphs: the first must use edges of a spanning subgraph 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 lower bound of $O(n\log n)$ for the number of edges (hence messages) used by any algorithm in the first class and a lower bound of $O(n^2)$ edges for the second class. 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 it's complexity is analyzed. This algorithm together with the lower bound of $O(n^2)$ for finding a minimum weight spanning tree presented in [6] 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 [1] - [8]; 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

### 2.1. Definitions and Axioms

In this section we study lower bounds for global algorithms and for matching algorithms (to be defined later). 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
$SEND = < send_1 , send_2 , \cdots , send_k >$
that includes all the events of the first type in their order of occurrence (if there are no such events then $SEND$ 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 $SEND(t) = < send_1 , \cdots , send_t >$ ($SEND(0)$ 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 of a sequence is not necessarily unique.

Let $NEW = NEW(SEND)$ be the subsequence $< new_1 , new_2 , \cdots , new_r >$ 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 from either side.) This means that the message $send_i = (v(send_i),e(send_i))$ belongs to $NEW$ if and only if $e(send_i) \neq e(send_j)$ for all $i > j$. $NEW(t)$ 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 *induced* 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*. Note that all the graphs $G(NEW)$ above have a fixed set $V$ of vertices (some of which may be isolated).

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 - defined below as axioms - hold for every algorithm $A$ and every graph $G$(*):

*axiom1* : the empty sequence is in $EX(A,G)$.

*axiom2* : 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 $NEW_1 o NEW_2$. ($NEW_1$ and $NEW_2$ do not *interfere* if no two edges $e_1$ and $e_2$ that occur in $NEW_1$ and $NEW_2$ respectively 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.)

*axiom3* : 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. However, if the algorithm $A$ is global then the following fact holds as well:

*axiom4* : if $NEW(t)$ is in $EX(A,G)$ and $C$ is a proper subgraph of $G(NEW(t))$ which is a union of some connected components, 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 $C$.)

(*) These axioms reflect only some properties of distributed algorithms which are needed here.

† In general, one expects $e(k)=e(U)$ for any subset $U$ of k vertices. However, the reader may construct simple algorithms for which $e(U_1) \neq e(U_2)$ for two distinct subsets $U_1$ and $U_2$ of equal cardinality. It is clear that such an algorithm must use the actual identities of the processors in the network.

### 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 $NEW$ in $EX(A,G)$ such that $G(NEW)$ has one connected component whose set of vertices is $U$ and the vertices in $V-U$ are isolated.

**Proof**: A desired sequence $NEW$ can be constructed in the following way. Start with the empty sequence (using *axiom* 1). Then 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 eventually 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 at least $O(n \log n)$.

**Proof**: For a subset $U$ of $V$ we define $e(U)$ to be the maximal length of a sequence $NEW$ 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

$$e(k) = \min\{\, e(U) \mid U \subseteq V, |U| = k \,\}^{\dagger}$$

Note that $e(n)$ is the edge complexity of the algorithm $A$.

The Theorem will follow from the inequality

$$e(2k+1) \geq 2e(k) + k + 1 \qquad (k < \frac{n}{2})$$

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 $NEW_1$ and $NEW_2$ 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 isolated), respectively. These two sequences do not interfere with each other, and therefore - by *axiom* 2 - their concatenation $NEW = NEW_1 o NEW_2$ is also in $EX(A,G)$. The proper subgraph $C$ of $G(NEW)$ satisfies the assumptions of *axiom* 4. Note that each node in $C$ has at least $k$ adjacent unused edges within $C$. By *axiom* 4 there is an extension of $NEW$ 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 *NEW* 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

$$e(n) \geq \frac{n+1}{2} log(\frac{n+1}{2}).$$

This implies the Theorem.

Q.E.D.

From this Theorem it follows that

**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 at least $O(nlogn)$.

**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 $O(nlogk)$ messages for global algorithms, where $k$ is the number of the initiators of the algorithm.

### 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* have a lower bound of $O(nlogn)$ 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 $\lfloor x \rfloor$ is the largest integer not larger than $x$).

**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 at least $O(n^2)$.

**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 *NEW* in the following manner:
Let $NEW_0$ be the empty sequence. For $i \geq 0$ if $G(NEW_i)$ does not contain a maximum matching, then $NEW_{i+1}$ is an extension of $NEW_i$ by a message $(v,e)$ where $e = (v,j)$ is chosen with smallest possible $j$ (we use here *axiom* 1, *axiom* 3 and the appropriate variant of *axiom* 4 for matching-type algorithms).

Eventually we construct in this way a sequence *NEW* in $EX(A,G)$ that does contain a maximum matching. 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 *NEW* which use an edge that connects $u_i$ or $v_i$ to some $j < u_i$. By the construction of *NEW* $n_i \geq u_i - 1 \geq i - 1$. Thus the length of *NEW* is greater than

$$0 + 1 + \cdots + (\left\lfloor \frac{n}{2} \right\rfloor - 1) = \frac{n^2}{8} + O(n).$$

(Note that we did not count the edges $(u_i,v_i)$ of the matching). This completes the proof of Theorem 3.

Q.E.D.

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 at least $O(n^2)$.

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

* 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 $n$.

In [4] it was noted that global algorithms in general graphs require $|E|$ 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 $|E|-1$ 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 $O(n^2)$ 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 his 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 $O(nlogn)$ for problems like finding a leader. We present now an algorithm of $O(nlogn)$ messages for this task. This algorithm can be used to design global algorithms of $O(nlogn)$ 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 (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 $O(nlogn)$ 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). The algorithm start by a *WAKE* message, received by any nonempty set of nodes.

During the algorithm, 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 $phase(i)$ of his king, which are updated during the execution of the algorithm. $status(i) = (phase(i),k(i))$ is called the *status* of node $i$. Before the algorithm starts $k(i) = identity(i)$ and $phase(i) = -1$ 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 king'doms, his kingdom.

A citizen, upon receiving a message, can delay it, ignore it, or transfer it to (or from) his king along a tree edge, or an edge connecting it to another king (which was already used by that king).

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

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 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 $phase(i) < phase(j)$ then king $j$ announces his new citizens that he is their new king, and if $phase(i) = phase(j)$ then he first increases his phase by 1 and then sends an appropriate updating message towards all his 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 him start his algorithm. At most one such message can reach any node.

(2) $ASK(phase(i),k(i))$ : this message is sent by king $i$ through an unused edge in an attempt to increase his kingdom, and might be transferred onwards by citizens. Each *ASK* message has a *status*, which is the status $(phase(i),k(i))$ of the king that originated it (in the time it was originated).

(3) *ACCEPT*(*phase*($j$)) : this message is sent by king $j$ in return to an *ASK* message from another king, telling him that he is willing to join his kingdom. (this message also might be transferred onwards by citizens.)

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

(5) *YOUR_CITIZEN* : this message is returned by a citizen upon receiving an *ASK* message originated by his own king.

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

### 3.4. The Algorithm for a King

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

*unused*($i$) denotes the set of all his unused edges, and initially contains all his $n-1$ adjacent edges. *father_edge*($i$) denotes the edge connecting $i$ to his father. *sons*($i$) denotes the set of edges connecting $i$ to his sons. *receive*($m$) means that if the list of received messages is not empty, then $m$ is the first message in it,and is taken out of the list (else the processor waits until he receives a message $m$). The algorithm for a king follows.

**The Algorithm for a King**

```
begin
phase(i) := -1; state(i) := king;
unused(i) := set of all adjacent edges;
sons(i) := Φ;
receive(m); [m will be either WAKE or ASK]
if m = ASK(phase(j),k(j))
   then state := CITIZEN
   else phase(i) := 0;
while (unused(i) ≠ Φ and state = KING) do
   begin
      choose e ε unused(i);
      send an ASK message along e;
      unused(i) := unused(i)-{e};
label: receive(m);
         [m will be one of the following:
         YOUR_CITIZEN,ACCEPT,ASK]
      case m of
         YOUR_CITIZEN : ;
            [do nothing and enter the while
            loop again]
         ACCEPT(phase(j)) :
            [let e be the edge that delivered this
            message]
            sons(i) := sons(i) ∪ {e}
            if phase(i) > phase(j)
            then
               send UPDATE(phase(i),k(i))
               along e;
            else [i.e., phase(i) = phase(j)]
               begin
                  phase(i) := phase(i) + 1;
                  send
                  UPDATE(phase(i),k(i))
                  to all
                  your sons [new and old]
               end;
         ASK(phase(j),k(j)) :
            if
            (phase(i),k(i)) > (phase(j),k(j))
            then goto label
            else state := CITIZEN
               [Sorry, you are no longer a king!]
      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.
```

### 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 his king. However, doing it in the straightforward way might use $O(n^2)$ messages. We incorporate some control mechanism into the algorithm, and reduce the number of messages to $O(n\log n)$. Beside the procedures and variables used by the algorithm for a king, we also use here the function *search*($x$) that fetches the first message of type $x$ from the list and takes it out of it (or waits for such a message otherwise). It can be used with several arguments; e.g., *search*(*ASK*,*UPDATE*) will fetch the first message that is either an *ASK* or an *UPDATE* message (or will wait for such a message otherwise).

After receiving an *ASK* message which he forwards towards his king (i.e., of status higher than his), a citizen $i$ has to remember - besides his status $(phase(i),k(i))$ - the status of this *ASK* message, which must be greater than $i$'s status. At this stage, $i$ waits for an *UPDATE* or *ACCEPT* message and he does not process any other *ASK* message. The processing of the received messages is done by the procedure *process_ask*, as follows:

On receiving an $UPDATE(phase(j),k(j))$ message (from his father), node $i$

1. updates his own status,
2. sends this message to all his sons,
3. compares his (new) status $a$ with the status $b$ of the last *ASK* message he had passed, and performs the following:
   - 3.1 if $a<b$ he continues to wait for a response *ACCEPT* for the *ASK* message,
   - 3.2 if $a=b$, and the *ASK* message was not received along a tree edge (i.e., it was received directly from the sending king), he returns through this edge a *YOUR_CITIZEN* message and waits for a new *ASK* message,
   - 3.3 if $a>b$ he waits for a new *ASK* message.

In 3.2 and 3.3 above $i$ discards the last *ASK* message he had passed, and exits the procedure *process_ask*.

On receiving an *ACCEPT* message from his father (in reply to the *ASK* message), he delivers it back through the appropriate edge, exits the procedure *process_ask*, and then waits for the corresponding *UPDATE* message (this part is done by the procedure *process_new_accept*).

Note that a citizen may receive an *ACCEPT* message along an edge which is not a tree edge (such a message must be a response to an *ASK* message originated by this citizen in those good old days when he still was a king). In such a case he adds this edge to his set of sons, and sends through it the last *UPDATE* message that he received (this part is done by the procedure *process_old_accept*). The algorithm for a citizen follows.

**The Algorithm for a Citizen**

```
procedure process_ask;
[you have just received a message m = ASK(phase(j),k(j)) along edge e]
begin
  if (phase(j),k(j)) > (phase(i),k(i))
  then
    begin
      send m to your father;
      while (phase(j),k(j)) > (phase(i),k(i))
      do begin
           m1 := search(UPDATE,ACCEPT);
           case m1 of
             UPDATE :
               begin
                 process_update;
                 if (phase(i),k(i)) = (phase(j),k(j)) and e is not a tree edge
                 then send YOUR_CITIZEN along e
               end;
             ACCEPT :
               [you have just received an ACCEPT(phase(j)) e along edge e']
           end [of the case statement]
         end [of the while loop]
    end [of the if statement]
  if (phase(j),k(j)) = (phase(i),k(i)) and e is not a tree edge
  then send YOUR_CITIZEN along e;
  [if (phase(j),k(j)) < (phase(i),k(i)) then the ASK message is ignored]
end [of process_ask]


procedure process_old_accept;
  [you have just received an ACCEPT(phase(j)) message along edge e' which is not a tree edge]
begin
  sons(i) := sons(i) ∪ {e'};
  send UPDATE(phase(i),k(i)) along e'
end [of process_old_accept]


procedure process_new_accept;
  [you have just received an ACCEPT(phase(j)) message along edge e' which is your father_edge;
  this ACCEPT must be a response to an ASK message you received along edge e]
begin
  sons(i) := sons(i) ∪ {e'};
  father_edge(i) := e;
  m2 := search(UPDATE);
  process_update;
end [of process_new_accept]


procedure process_update;
  [you have just received an UPDATE(phase(j),k(j)) message along edge e' which is your father_edge]
begin
  phase(i) := phase(j); [phase(i) is increased by at least one]
  k(i) := k(j);
  send UPDATE(phase(i),k(i)) to all your sons
end [of process_update]
```

```
begin [of the main program for a citizen]
    [you have just received an ASK(phase(j), k(j))
    message (which changed your status from king to
    citizen) along some edge e.]
    if e ε sons(i) then sons(i):=sons(i)−{e};
    father_edge(i) := e;
    send ACCEPT(phase(i)) along the edge e;
    m:=search(UPDATE);
        [now m = UPDATE(phase(j), k(j));m was
        sent along e] phase(i) := phase(j);
        [phase(i) is increased by at least one]
    k(i) := k(j);
    send UPDATE(phase(i), k(i)) to all your sons;
    repeat
        receive(m);
            [m, received along e', will be one of the fol-
            lowing:
            ASK, UPDATE, ACCEPT, LEADER]
        case m of
            ASK(phase(j),k(j)) :   process_ask;
            UPDATE(phase(j),k(j))
            process_update;
            ACCEPT(phase(j)):
                if e' is not a tree edge
                then process_old_accept
                else process_new_accept;
        end [of the case statement]
    until m =LEADER;
end.
```

#### 3.6. 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 5:** In any execution of the algorithm, eventually only one king remains.

**Proof:** Assume that the the Theorem is false. Since it is impossible to have no king, the following must hold:

(FA): In some execution of the algorithm, $s > 1$ nodes remain kings forever. Denote them $king_1, \ldots, king_s$, and suppose that $status(king_i) < status(king_j)$ for $i < j$.

The proof proceeds by three lemmas.

**Lemma 2:** Under the assumption (FA), eventually every node in the network will have his status equal to $(x, king_i)$ for some $1 \le i \le s$.

**Proof:** Otherwise, some node $j$ has a different status $(phase(j),k(j))$ forever. This must be a status that he received from some king $t$ that is now a citizen ($t$ may be equal to $j$). $t$'s status was changed when he became a citizen. At this point he sent an *ACCEPT* message that eventually was answered by an *UPDATE* message. This *UPDATE* message contained a status with a new king, and eventually reached $j$. Clearly, $t$ never again became a king, which contradict the assumption that $k(j) = t$. □

**Lemma 3:** Under the assumption (FA), if for some $1 \le i \le s$ $king_i$ is not asleep, then he eventually sends an *ASK* message to a node not in his kingdom.

**Proof:** Suppose $king_i$ sends his first *ASK* message at his final phase to a node $j$ in his kingdom. By Lemma 2, node $j$ eventually knows that his status is equal to $status(king_i)$ and will send him back a message *YOUR_CITIZEN*. $king_i$ will now send an *ASK* message along some other unused edge. **Since the underlying graph is complete,** this process continues until an *ASK* message is sent outside $i$'s kingdom. □

**Lemma 4:** Suppose $king_s$ sends an *ASK* message to a node $a$ in $king_j$'s kingdom ($j < s$). Then $king_j$ will eventually become a citizen.

**Proof:** If $a = king_j$ then he will become a citizen of $king_s$ immediately after receiving the message. Otherwise, this *ASK* message either arrived at $king_j$ or was stopped somewhere on the way between $a$ and $king_j$ (it cannot be discarded, since its status is greater then that of $king_j$). In the second case some other *ASK* message of status higher than $status(king_j)$ was forwarded towards $king_j$ by the node that blocked $king_s$'s message. Applying this reasoning as long as needed, we conclude that an *ASK* message of a status higher than $status(king_j)$ will eventually reach $king_j$. At this point $king_j$ will become a citizen. □

By Lemma 4 we get a contradiction to the assumption that $s > 1$, and this completes the proof of the Theorem.

Q.E.D.

**Corollary:** The unique remaining king eventually announces his leadership (and the algorithm stops).

**Proof:** By a proof similar to the one of Lemma 3, it can be shown that this king will eventually exhaust all his unused edges by sending *ASK* messages and receiving *YOUR_CITIZEN* replies. When no unused edges remains, he will send the *LEADER* message to all the nodes in the network, and each node, upon receiving this message, will stop his algorithm. □

#### 3.7. Complexity Analysis of the Algorithm

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

**Theorem 6:** 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 5:** If $k$ nodes start the algorithm by *WAKE* messages and node $i$ is the leader, then when the algorithm stops we have

$$phase(i) \leq \lfloor log_2 k \rfloor .$$

**Proof:** Whenever a king at phase $t$ increases his phase, he annexes another king of phase $t$. Therefore, we have at most $\frac{k}{2}$ kings in phase 1, $\frac{k}{2^2}$ kings in phase 2, ..., $\frac{k}{2^l}$ kings in phase $l$, for every $1 \leq l \leq \lfloor log_2 k \rfloor$.

□

**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) *YOUR_CITIZEN* : each node sends at most one such message - as a reply to an *ASK* message - per phase. Therefore, the total number of such messages is bounded by $nlog_2 k$.

(4) *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, therefore the total number of such message sent by kings is bounded by $nlog_2 k$. 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 $nlog_2 k$.

(5) *ACCEPT* : the total number of such messages sent by a king (during the algorithm) is $k-1$. the total number of such messages sent by all citizens is not larger than the total number of *ASK* messages sent by all citizens, hence it is also bounded by $nlog_2 k$.

(6) *UPDATE* : each citizen receives at most one such message per phase, hence the total number of such messages is also bounded by $nlog_2 k$.

To conclude, the total number of messages used by the algorithm does not exceed $5nlog_2 k + O(n)$.

Q.E.D.

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, *loglogn* bits for the phase, and *logm* bits for the identity (where $m$ is the largest identity). Therefore the maximal number of bits per message is bounded by $\log_2[4mlog_2 n]$.

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

## REFERENCES

[1] J. E. Burns, *A formal model for message passing systems* , TR-91, Indiana University, September 1980.

[2] D. Dolev, M. Klawe and M. Rodeh, *An O(nlogn) unidirectional distributed algorithm for extrema finding in a circle* , J. of Algorithms, 3, 1982, pp. 245-260.

[3] R. G. Gallager, *Choosing a leader in a network* , unpublished memorandum, M.I.T.

[4] R. G. Gallager, P. A. Humblet and P. M. Spira, *A distributed algorithm for minimum spanning tree* , Transactions on Programming Languages and Systems, 5, 1, 1983, pp. 66-77.

[5] D. S. Hirschberg and J. B. Sinclair, *Decentralized extrema- finding in circular configurations of processes* , Comm. ACM 23 (November 1980).

[6] E. Korach, S. Moran and S. Zaks, *Finding a Minimum Spanning Tree can be harder than finding a spanning tree in a distributed network* ,TR #294,Dept. of Computer Science, Technion, Israel (October 1983).

[7] J. Pachl, E. Korach and D. Rotem, *Lower bounds for distributed maximum-finding algorithms* , to appear in The Journal of the ACM.

[8] G. L. Peterson, *An O(nlogn) unidirectional algorithm for the circular extrema problem* , Transactions on Programming Languages and Systems, 4, 1982, pp. 758-762.

[9] N. Santoro, *On the message complexity of distributed systems*, SCS-TR-13, School of Computer Science, Carleton University, Ottawa, Canada (1982).