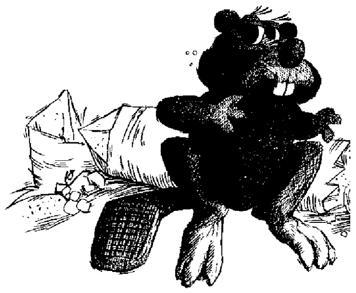

Distributed Ranking

Ephraim Korach
Doron Rotem
Nicola Santoro

Technical Report
CS-83-22

July, 1983

DISTRIBUTED RANKING

Ephraim Korach

Doron Rotem

Nicola Santoro

Technical Report CS-83-22

ABSTRACT

Ranking refers to the process of determining the rank of n uniquely numbered processors loosely coupled in an asynchronous network where no central controller exists and the number of processors is not known a priori. We present efficient algorithms for the ranking process: these algorithms are fully distributed and, in order to be executed, require only local topological knowledge at each processor. The message complexity of the algorithms is discussed both in the worst and average case.

Index Terms

Distributed algorithms, communication complexity, ranking, graph theory.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper has been presented at the 13th SE Conference on Combinatorics, Graph Theory and Computing, 1982. E. Korach is with the IBM Research Center, Haifa, Israel. D. Rotem is with the Department of Computer Science, University of Waterloo, Waterloo, Canada. N. Santoro is with the School of Computer Science, Carleton University, Ottawa, Canada.

1. INTRODUCTION

Consider a network of n loosely-coupled processors which work asynchronously and communicate by exchanging messages with their neighbours in the network. All processors have the same program and differ only by having a distinct value (known only to the owner) stored in their local (non-shared) memory. There is no central controller; no processor has a priori knowledge of the number of processors in the network; and each processor is only aware of its neighbours. Processing time is assumed to be negligible when compared to transmission and queuing delays.

Recently, great attention has been given to the problem of determining, in such an environment, the smallest or the largest value in the network (known as election' or 'extrema-finding' problem). The goal is to obtain an efficient solution algorithm, where the efficiency is measured in terms of number of message exchanges between neighbouring processors and/or of time units needed to complete the execution of the algorithm. Solution algorithms have been presented for the case when the network is (known to the processors to be) a tree [2,11], a unidirectional circle [6,18,19], a bidirectional circle [1,3,5,7,12,15,17,23], a complete graph [13], and for the case when nothing is known to the processors about the network topology (the 'arbitrary graph' case) [9]. The related problem of selecting the k-th smallest or largest value has also been investigated, and solution algorithms have been developed for bidirectional circles [15], trees and meshes [8], and fully connected networks [20,21,25,26]; in the latter results more than one value is allowed to be stored at any one processor.

In this paper, we investigate the related problem of determining the rank of all values in the network, called the *ranking* problem. Ranking, which can be thought of as the distributed version of sorting, is important in situations where the control token is assigned sequentially to the processors according to the rank of some associated value (e.g., amount of available resources, amount of locally originated requests, etc.) [16,17]; furthermore, any ranking algorithm can be obviously used to solve the selection problem.

The paper is organized as follows. In the next section the framework for which the results are established is formally described. A ranking algorithms for trees is presented in section 3, and its correctness is proved. In section 4, the worst and average case complexity of the proposed algorithm is analyzed and, in section 5, it is shown how this algorithm can be efficiently employed for ranking in the case where the processors have no apriori knowledge of the network topology. Finally, some extensions and open problems are briefly discussed in section 6.

2. THE FRAMEWORK

The network under consideration is a couple (G, μ) where $G = (N, A)$ is a connected undirected graph with no self-loops or multiple edges, and $\mu : \{1,2,\ldots,|N|\} \to N$ is a bijection called ranking function. Every node $x \in N$ represents a processor in the network and has associated with it a distinct value; for simplicity, we shall denote by i the node whose corresponding processor has value i. Each edge $(x, y) \in A$ represents a direct bidirectional link between nodes x and y. The ranking function μ defines a total ordering $>_\mu$ on the values in the network of which all processors are aware.

If $(x, y) \in A$, x and y are said to be neighbours; each processor i maintains a list L_i of its neighbours and can send a message 'M' to a subset $S \subseteq L_i$ of its neighbours by the " *send M to S*" directive. Each processor has a queue in which incoming messages are stored and processed in first come first serve order; moreover, the communication links (represented by the edges) preserve the order of the messages sent. Local processing time is assumed to be negligible when compared to transmission and queueing delays; that is, operations are performed at a processor instantaneously. In addition to the standard graph-theoretical terminology, we use the following notation:

$d_G(x, y)$ distance (i.e., length of shortest path) between x and y in the graph G, and for a tree T, $T_x(y)$ is the largest subtree containing y in the forest obtained by removing from T the node x and its incident edges (see Fig. 1).

3. RANKING IN TREE NETWORKS

In this section, we consider the case where G is a tree and all processors are aware of it, and present an algorithm which ranks the values exchanging $.5n^2 + 0(n)$ messages in the worst case and $n^{1.5} + 0(n)$ messages in the average case. One or several processors can independently start the execution of the algorithm upon reception of a locally originated request. The algorithm is expressed in terms of what operation a processor $i \in N$ must perform when it is in a given state upon certain conditions; e.g., reception of a particular type of message. Initially, all processors are in state *'available'*. Each processor i has available to it a table v_i where the entry $v_i(y)$, $y \in L_i$, is used to store the values received from y; in addition, there is a state variable $\bar{v}_i$ used to store the processor's own value. By convention, $v_i(y) = \delta$ means that no (unranked) value is known from $T_i(y)$, and $v_i(y) = \infty$ means that all values in $T_i(y)$ have already been ranked.

Algorithm 3.1 is composed of two phases: Initialization and Ranking. In the Initialization phase, all processors are activated and the leaf processors start a process of determining the smallest known value; at the end of the phase, one processor (say, r) enters a special state called *'ranking'*, and every processor y knows the smallest value in each subtree $T_y(z)$ were $z \in L_y$ except for the tree $T_y(r)$ (i.e., the subtree which contains the *'ranking'* processor r). In the

Ranking phase, the values are sequentially ranked; this process is started by r. Upon completion, the execution of the algorithm terminates at all nodes. A formal description of Algorithm 3.1 is contained in the Appendix.

In order to prove the correctness of the algorithm, let us analyze each phase in detail.

Initialization

The Initialization phase is composed of three stages: activation, set-up and resolution. One or several processors may independently start the activation stage (and, thus, the execution of the algorithm) by becoming *'active'* and sending an A(activation) message to their neighbours. When an *'available'* processor receives an A message, it becomes *'active'* and activates its neighbours. It is not difficult to see that, within finite time from the start of this stage, all processors will have been activated.

The set-up stage is started by the leaf processors (i.e., processors with only one neighbour) as soon as they become *'active'*; since all processors eventually become *'active'*, all leaf processors will start the set-up stage, although not simultaneously. To start this stage, an *'active'* leaf processor i will 1) send a S(set-up) message containing its value i to its (only) neighbour, say x; 2) set $v_i(x) := \delta$; and 3) enter state *'processing'* - An *'active'* non-leaf processor i, upon reception of an S message containing value j from neighbour $x \in L_i$, will set $v_i(x) := j$; when i has received an S message from all neighbours but one, say z, it will 1) send a S message containing the smallest known value to z, 2) set $v_i(z) := \delta$, and 3) enter state *'processing'*. When a *'processing'* node receives an S message, it becomes *'saturated'* and starts the resolution stage.

The activation and set-up stages constitute a mechanism similar to the one employed in other distributed algorithms (e.g. [4,14,22]); the S messages are 'filtered' through the network and collected at the *'saturated'* processors. Following, is a well-known property of this mechanism [4,14] expressed here in terms of the processor states used in this paper.

Property 3.1. One or two neighbouring processors become *'saturated'*. ▪

In the case of two *'saturated'* processors, since they are neighbours, each must have entered such a state upon reception of an S message sent by the other. The resolution stage is started and performed only by these two nodes. At the end of this stage, which does not involve the transmission of any message, one of the *'saturated'* nodes, say j, ignores the received message and becomes *'processing'*; the other, say i sets $v_i(j)$ to the received value, and enters the state *'ranking'*. This processor starts the execution of the next phase of the algorithm that is

Property 3.2. Exactly one processor starts the Ranking phase. ▪

Summarizing, all leaf processors send their value toward the rest of the tree and become *'processing'*; each non-leaf processor collects these values from all neighbours but one, sends the smallest known value to such a neighbour, and becomes *'processing'*. That is, within finite time from the start of the algorithm, all processors must have become *'processing'*. Since an 'S' message only carries the smallest value known to the sender, we have

Property 3.3. Let i be *'processing'* in the Initialization phase. Then

(i) there is only one $x \in L_i$ such that $v_i(x) \neq \delta$;

(2) $\bar{v}_i = i$;

(3) for all $x \in L_i$, if $v_i(x) \neq \delta$ then $v_i(x)$ is the smallest value in $T_i(x)$. ▪

Finally, one *'processing'* processor receives an S message from the only unacknowledged neighbour (i.e., the neighbour x with $v_i(x) = \delta$) and becomes *'ranking'*. Hence

Property 3.4. Let i start the Ranking phase. Then i knows the smallest value in the network. ▪

Ranking

In the Ranking phase, all values are ranked successively starting from the smallest one. Initially, a counter c is set to one by the first *'ranking'* processor.

A ranking processor i determines the smallest unranked value m in the network. If $m \neq i$ (i.e., its value is not the one to be ranked next), it determines the neighbour $des \in L_i$ for which $v_i\,(des) = m$ and sends to des a R(ranking) message with a triple (c, m, s) where $s = Min(\{v_i(y) : y \in L_i - \{des\}\} \cup \{\bar{v}_i\})$ is the second smallest unranked value known to i, and c is the next rank to be assigned. It then sets $v_i(des) := \delta$ and becomes either *'processing'* or *'terminating'*, depending on whether $s \neq \infty$ or $s = \infty$, respectively.

If $m = i$ (i.e., its value is the next one to be ranked), it ranks itself by c, increases c by one, sets $\bar{v}_i := \infty$, and determines the new smallest m and second smallest s unranked values known to it. As proved later, this value m is actually the smallest unranked value in the entire network; therefore, if $m = \infty$ (i.e., all values have been ranked), the processor becomes *'terminating'*, otherwise it performs the process described above for the case $m \neq i$.

When a *'processing'* processor j receives an R message containing (c, m, s) from a neighbour x, it sets $v_i(x) := s$, becomes *'ranking'* and performs the process described above.

Observe that a *'processing'* processor becomes *'ranking'* only upon reception of an R message. At the same time, an R message is sent only by a *'ranking'* processor and only to one neighbor; furthermore, the *'ranking'* processor changes state after doing so. Therefore, by Property 3.2, we have

Property 3.5. At any time during the execution of Algorithm 3.1 there is at most one *'ranking'* processor. ▪

We will now show that a *'ranking'* processor knows, while it is in state *'ranking'*, the smallest unranked value in the network.

Property 3.6. Let i become *'ranking'*. Then, for all $j \in N$ and all $y \in L_j$ we have

(1) if $i \in T_j(y)$ then $v_j(y) = \delta$.

(2) if $i \notin T_j(y)$ then $v_j(y)$ is the smallest unranked value in $T_j(y)$.

Proof: By induction. By Properties 3.2 and 3.3, the claim holds when the first processor becomes *'ranking'*. Let the claim hold when j becomes *'ranking'* and let i be the processor which becomes *'ranking'* next; we will show that the claim holds also when i becomes *'ranking'*. Observe that the only two processors which will modify their tables and/or state variables are the processors i and j; therefore, we must only show that the claim holds for i and j when i becomes *'ranking'*. By the induction hypothesis, for all $x \in L_j$, $v_j(x)$ is the smallest unranked value in $T_j(x)$; hence, $m = Min(\{v_j(y) : y \in L_j\} \cup \{\bar{v}_j\})$ is the smallest unranked value in the network. We shall consider two cases, depending on whether $m \neq j$ or $m = j$, respectively.

Case 1 ($m \neq j$): Processor j sends an R message containing (c, m, s) to the neighbour i, where c is the next rank to be assigned, and s is the second smallest unranked value known to j. Processor j then sets $v_j(i) := \delta$ and becomes either *'processing'* or *'terminating'* depending on whether $s \neq \infty$ or $s = \infty$, respectively. Hence, when i becomes *'ranking'* upon reception of this R message sent by j, the claim holds for j. Let us show that the claim holds also for i; that is, once i becomes *'ranking'* for all $x \in L_i$, $v_i(x)$ is the smallest value in $T_i(x)$. The only change in the Table v_i occurs in the entry $v_i(j)$ which is modified from δ to s. By the induction hypothesis, s is the smallest unranked value in $T_i(j)$; hence, the claim holds.

Case 2 ($m = j$): Processor j ranks its value by the current value of counter c and sets $\bar{v}_j := \infty$. Since no other value in table v_j is changed and since, by the induction hypothesis, $v_j(x)$ is the smallest unranked value in $T_j(x)$, for all $x \in L_j$ then, $m = \min\{v_j(y) : y \in L_j\}$ is the new smallest unranked value in the network. Furthermore, now $m \neq j$; therefore, by Case 1, the claim holds. ▪

As a consequence of Property 3.6, when a processor becomes *'ranking'*, it knows the smallest unranked value in the network. Every *'ranking'* processor i sends the R message containing (c, m, s) towards its final destination m; since the values of c and m are not modified by any of the processors on the path from i to m, m will rank its value by c. In the proof of Case 2 of Property 3.6, it has been shown that, after a processor ranks its value, it still knows the smallest unranked value in the network; furthermore, it increases the value of counter c by one. Therefore

Lemma 3.1. Every processor correctly ranks its value. ▪

Let us now discuss the termination of the algorithm. First observe that a processor can become *'terminating'* only from state *'ranking'*. Let j be *'ranking'* and $i \in L_j$ be the next *'ranking'* processor; then j becomes *'terminating'* if and only if $\bar{v}_j$ and all entries in v_j, except $v_j(i)$, are ∞ (i.e. $s = \infty$). By using Lemma 3.1 and observing that $\bar{v}_j$ is set to ∞ when j ranks its value, and that $v_i(j)$ is set to s upon reception at i of the R message from j, it is not difficult to see that the following lemma holds.

Lemma 3.2. Every processor terminates its execution of the algorithm. ▪

Therefore, by Lemmas 3.1 and 3.2, we have shown the following

Theorem 3.1. Algorithm 3.1 correctly and within finite time solves the ranking problem in a tree. ▪

4. COMPLEXITY ANALYSIS

A. Worst Case Message Complexity

Let T_n denote the set of all couples (T, μ) where $T(N, A)$ is a tree with $|N| = n$ nodes, and $\mu : \{1,\ldots,n\} \to N$ is a ranking function; that is, $\mu(i)$ is the node whose value has rank i.

The worst case message complexity of Algorithm 3.1 is

$$W(n) = Max(\{f(T, \mu) + g(T, \mu) ; (T, \mu) \in T_n\})$$

where $g(T, \mu)$ is the number of messages exchanged until $\mu(1)$ is ranked, and

$$f(T, \mu) = \sum_{i=1}^{n-1} d_T(\mu(i), \mu(i+1))$$

where $d_T(x, y)$ is the distance between x and y in T.

Lemma 4.1. Let $(T, \mu) \in T_n$. Then

$$f(T, \mu) \leq 4n - 3$$

Proof: During the activation stage, exactly two A messages are exchanged on each edge; during the set-up stage, each processor sends only one S message; and no message is exchanged during the resolution stage. Therefore, at most $2(n-1) + n = 3n - 2$ messages are exchanged in the Initialization phase. When the first processor, say r, becomes *'ranking'*, an R message is sent through the path from r to $\mu(1)$; hence, an additional $d_T(r, \mu(1))$ messages will be exchanged. Since $d_T(r, \mu(1)) \leq n - 1$, the Lemma holds. ▪

Lemma 4.2. Let P be a simple path of $n - 1$ edges. Then for all $(T, \mu) \in T_n$

$$f(T, \mu) \leq f(P, \mu)$$

Proof: Given a couple $(T, \mu) \in T_n$, assume T contains a node x of degree $k \geq 3$. Let $T^1, T^2, \ldots, T^k$ denote the trees obtained by removing from T the node x and its incident edges (see Figure 2). Let us denote by m_{ij} $(1 \le i \ne j \le k)$ the set of pairs of processors with consecutive ranks $(\mu(i), \mu(i+1))$ such that one member of each pair is in T^i and the other is in T^j. Each pair in m_{ij} represents path which is traversed by an R message. Without loss of generality, let m_{12} be the set of largest cardinality, i.e., $|m_{12}| = Max\ \{|m_{ij}| : 1 \le i \ne j \le k\}$. We now construct a tree T' from T by disconnecting T^2 and appending it to a leaf at the end of a longest root to leaf path of T^3, the length of this path is $h(T^3)$, the height of T^3. Comparing T' with T, each pair of m_{12} which represented a path of length x in T represents a path of length $x + h(T^3)\ + 1$ in T', a pair of m_{23} which represents a path of length y in T represents a path of length at least $y - (h(T^3)\ + 1)$ in T'. All other pairs are unchanged or represent longer paths in T'. The net change is at least $(|m_{12}| - |m_{23}|)(h(T^3) + 1) \ge 0$ since $|m_{12}| \ge |m_{23}|$. This implies that every tree T which is not a simple path, can be reduced to a simple path P in which algorithm 3.1 requires at least as many messages. ∎

As a consequence of Lemma 4.2, we can restrict our analysis of $f(T, \mu)$ to the case of T being a simple path. Let

$$f(n) = Max(\{f(T, \mu) : (T, \mu) \in T_n\})$$

Lemma 4.3.

$$f(n) = \begin{cases} \frac{n^2}{2} - 1.5 \text{ if } n \text{ is odd} \\ \frac{n^2}{2} - 1 \text{ otherwise} \end{cases}$$

Proof: Let T be a simple path, and let $l_1, l_2, \ldots, l_{n-1}$ be the edges of T from left to right. The edge l_i divides T into a left and a right subtree of i and $n-i$ vertices respectively. Each message which passes along l_i represents a pair of consecutive ranks one in the left and one in the right subtree. Hence on the edge l_i, at most $2\ Min(\{i, n-i\})$ messages will be transmitted under any bijection μ. Also, at least one edge, say l_k, has either $\mu(1)$ or $\mu(n)$ in its smallest subtree; and on l_k at most $2 \cdot Min(\{k, n-k\}) - 1$ messages will be transmitted. Therefore, at most

$$2\left(\sum_{i=1}^{n-1} Min(\{i, n-i\})\right) - 1$$

messages will be transmitted during the Ranking phase.

For n even, we have

$$2\sum_{i=1}^{n-1} \min\{i, n-i\} = 2\sum_{i=1}^{n/2} i + 2\sum_{i=1}^{n/2-1} i = \frac{n}{2}\left(\frac{n}{2}+1\right) + \left(\frac{n}{2}-1\right)\frac{n}{2} = \frac{n^2}{2}$$

Hence,

$$f(n) \le \frac{n^2}{2} - 1$$

To prove the equality in the above relation, consider the following bijection μ.

Let $n = 2 \cdot k$ for some positive integer k, and let us call the nodes $x_n, x_{n-1} \cdots x_1$

$$\mu^{-1}(x_i) = \begin{cases} 2i & 1 \le i \le k \\ 2i - (n+1) & k+1 \le i \le n \end{cases}$$

Then, in this case (see Figure 3(a))

$$f(T, \mu) = 2 + k + \sum_{i=1}^{k} (2i+1) = 2k^2 - 1 = n^2/2 - 1$$

Similar arguments show that for n odd we have

$$f(n) \le \frac{n^2}{2} - 1.5$$

and that the bound is achieved by the bijection defined below, where $n = 2k + 1$

$$\mu^{-1}(x_i) = \begin{cases} n - 2i & 1 \le i \le k \\ n & i = k+1 \\ 2n + 2 - 2i & k+1 < i \end{cases}$$

(see Figure 3(b)) and the Lemma is proved. ▪

Therefore, by Lemmas 4.1 and 4.3, we have shown the following

Theorem 4.1. $W(n) \le \frac{n^2}{2} + 4n - 4$ ▪

B. Average Case Message Complexity

Let us now analyze the average-case (message) complexity of the algorithm. It is known that

$$|T_n| = n^{n-2}$$

Assuming that each element of T_n is chosen with equal probability the average-case (message) complexity of Algorithm 3.1 is

$$A(n) = \frac{1}{n^{n-2}} \sum_{(T,\mu) \in T_n} (f(T,\mu) + g(T,\mu))$$

Let E_k be the set of all edges, in the trees of T_n, which connect a subtree of k processors to a subtree of $n-k$ processors. Clearly, $E_k = E_{n-k}$ by this definition, and since each tree contains $n-1$ edges

$$\sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} |E_k| = (n-1)\, n^{n-2}$$

Lemma 4.3.

$$|E_k| = \begin{cases} \binom{n}{k} |T_k| \; |T_{n-k}| \, k(n-k) \text{ if } k \ne \frac{n}{2} \\ \frac{1}{2} \binom{n}{k} |T_k| \; |T_{n-k}| k(n-k) \text{ otherwise} \end{cases}$$

Proof: Let $k \ne n/2$. There are $\binom{n}{k}$ ways to assign ranks from $\{1,...,n\}$ to the nodes of a tree of size k; for each such rank assignment, there are $|T_k|$ trees which can be formed accordingly and $|T_{n-k}|$ trees where the remaining ranks can be assigned. Every pair of trees T' and T'', with k and $n-k$ nodes respectively, can be connected by an edge in $k(n-k)$ ways to form a tree with n nodes, and the connecting edge is a member of E_k. It is not difficult to see that all connecting edges counted in this way are distinct and that the entire set E_k is enumerated in this way. Hence, the lemma holds.

Let $k = n/2$. The above argument still holds; however, each edge is now counted twice and hence the factor $1/2$ is introduced. ▪

Lemma 4.4. The expected number of messages transmitted along an edge $e \in E_k$ in the Ranking phase is $2k(n-k)/n$.

Proof: For any pair of consecutive ranks i, $i+1$, the probability that a message will pass along $e \in E_k$ is

$$\frac{2k\,(n-k)}{n\,(n-1)}$$

This follows since $\mu(i)$ and $\mu(i+1)$ must be in the trees T' and T'' of size k and $n-k$, respectively, which are connected by e. The multiplicative constant 2 comes from interchanging $\mu(i)$ and $\mu(i+1)$. Since there are $(n-1)$ possible consecutive pairs of ranks, the lemma holds. ▪

Theorem 4.2.

$$A(n) = \frac{1}{n \cdot n^{n-2}} \sum_{k=1}^{n-1} \binom{n}{k} \, |T_k| \, |T_{n-k}| \, k^2(n-k)^2 + 0(n)$$

Proof: By definition of $A(n)$ and by Lemma 4.1, we have

$$A(n) = \frac{1}{n^{n-2}} \sum_{(T,\mu)\in T_n} f(T,\mu) + \frac{1}{n^{n-2}} \sum_{(T,\mu)\in T_n} g(T,\mu) =$$

$$= \frac{1}{n^{n-2}} \sum_{(T,\mu)\in T_n} f(T,\mu) + 0(n)$$

To obtain the first term in the expression above, we sum the expected number of messages in E_k, for k between 1 and $n-1$. Since for $k \neq \frac{n}{2}$, $E_k = E_{n-k}$ and by using Lemma 4.3 for $k = \frac{n}{2}$, it follows that this sum must be divided by two and the claimed result is obtained. ▪

By using Stirling approximation formula and the asymptotic formula

$$\sum_{k=1}^{n-1} \frac{1}{\sqrt{k(n-k)}} = \pi + 0(n^{-\frac{1}{2}})$$

given in [10], we can derive the following asymptotic expression for $A(n)$

$$A(n) = \frac{n^{\frac{3}{2}}\sqrt{\pi}}{\sqrt{2}} + 0(n)$$

C. Time Complexity and Message Size

Another important measure of efficiency is the total time delay; that is, the delay between the time the first processor starts the execution of the algorithm and the time the last processor terminates its execution. Since the network is asynchronous and transmission and queuing delays are unpredictable, the total time delay may vary for different executions of the same algorithm on the same network. However, an estimate can be obtained by measuring the 'ideal' or 'minimal' time delay; that is, the total time delay experienced when the network behaves synchronously and transmission delays are unitary. Note that the number of time units expressed by the ideal time delay corresponds to the number of 'beats' in a parallel computation.

It is not difficult to see that the Ranking phase of Algorithm 3.1 is strictly sequential, while the Initialization phase is fully parallel. Hence, the number of time units required by the algorithm in the worst and in the average case will be dominated by the ideal time delay encountered in the Ranking phase. That is, $T_{av}(n) = 0(A(n))$ and $T_{wo}(n) = 0(W(n))$.

Finally, the time and message complexity of a distributed algorithm depend on the size of the messages. In all *'election'* algorithms, the number of fields in a message is a (small) constant independent of n, and each field contains either an identifier, or a processor value, or $0(\log n)$ bits. It is easy to observe that we use the same metric; in fact, Algorithm 3.1 requires messages composed of at most four fields (e.g., the R messages), each containing either an identifier (e.g., A,S,R,T) or a processor value, or $\lceil \log_2 n \rceil$ bits (to send the rank r). Therefore, the complexity of our algorithm is evaluated in the same metric space used to measure the efficiency of the *'election'* algorithms.

5. RANKING IN ARBITRARY GRAPHS

In this section, we consider the case where nothing is known to the processors about the network topology; i.e., the arbitrary graph case [2,4,11,14].

As in other distributed problems, solution algorithms for arbitrary graphs can be obtained by first constructing a spanning-tree of the graph and then applying on this tree a solution algorithm for trees (e.g. [4,9,14]). Note that the construction of a spanning-tree in our framework does not require the determination of global information (e.g., the adjacency matrix of the tree) at the processors; instead, it only requires the determination at each processor of which neighbours are also neighbours in the tree. Furthermore, the spanning-tree construction can be done in parallel with the execution of a solution algorithm for trees.

A problem with this approach lies in the fact that several (possibly all) processors start the execution of the algorithm independently of each other; that is, several spanning-trees may be concurrently constructed increasing dramatically the message complexity of the resulting algorithm. This problem can be *'by-passed'* by first executing an 'election' algorithm for arbitrary graphs (e.g., [9,11]) and then letting only the 'elected' processor start the construction of the spanning-tree and the execution of the ranking algorithm for trees; in this way, only one spanning-tree will be constructed and, since all processors are in that tree, they will correctly rank their values using Algorithm 3.1 in the spanning tree.

Let us now describe the ranking algorithm for an arbitrary graph.

Algorithm 5.1.

1. Execute an *'election'* algorithm

2. If 'elected' start the construction of a spanning-tree and the execution of Algorithm 3.1; otherwise, participate in the construction of the spanning-tree and in the execution of Algorithm 3.1 when notified.

Since an *'election'* can be performed in an arbitrary graph transmitting at most $O(e) + O(n \log n)$ messages [9], where e is the number of edges in the graph, and since the construction of a spanning-tree with only one initiator requires an additional $O(e)$ messages [14,22], from Theorem 4.1, we have

$$W(n, e) \leq O(n^2)$$

where $W(n, e)$ denotes the number of message exchanges needed in the worst case to rank all values in networks with n nodes and e edges by Algorithm 5.1.

The time complexity of Algorithm 5.1 is dominated, as in Algorithm 3.1, by the Ranking phase and we have

$$T_{wo}(n, e) = O(W(n, e))$$

where T_{wo} denotes the worst-case number of time-units required by Algorithm 5.1. Also in this case, the time and message complexities are evaluated in the same metric space used by the *'election'* algorithms.

6. CONCLUDING REMARKS

In the previous sections, we have presented ranking algorithms for trees and arbitrary graphs. For other fixed topologies of some practical interest, ad-hoc algorithms can be devised. In particular, if G is a complete graph, ranking can be done by first executing an election algorithm which finds a 'leader' and then letting the leader obtain all values, rank them, and finally notify all processes of their result. The election can be done exchanging $O(n \log n)$ messages [13], and the ranking can be completed with an additional $O(n)$ messages. In the case that G is a bidirectional circle, Algorithm 5.1 will rank all values in

$0(n^2)$ messages and time units; this result cannot be extended to unidirectional circles. A different algorithm which can be used in both unidirectional and bidirectional circles is described in [24]; this algorithm requires $0(n^2)$ messages and only $0(n)$ time units.

The proposed algorithms can obviously be employed to select the k-th smallest (or largest) value by stopping their execution as soon as such value has been ranked. By simple modifications to the algorithms, we can select the k-th smallest element in a tree in $0(n \cdot Min(\{k, n+1-k\}))$ messages; therefore, selecton can be performed in arbitrary graphs in $0(e) + 0(n(log\ n + Min(\{k, n+l-k\})))$ messages.

Finally, the following problems are still open: to establish a lower-bound for the time and message complexity of the ranking problem; to establish upper and lower bounds on the average case time and message complexity for arbitrary graphs; to improve the upper-bounds presented in this paper.

REFERENCES

[1] M.F. Aburdene, "Distributed algorithms for extreme-finding in circular configurations of processors", in *Proc. IEEE Int. Symp. Large Scale Systems*, Oct. 1982.

[2] D. Angluin, "Local and global properties in networks of processors", in *Proc. 12th ACM Symp. Theory of Comput.*, May 1980.

[3] J. Burns, "A formal model for message passing systems", Indiana Univ. Tech. Rep. TR-91, 1980.

[4] E.J. Chang, "Echo algorithms: depth parallel operations on general graphs", *IEEE Trans. Software Eng.*, vol. SE-8, July 1982.

[5] E.J. Chang and R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configurations of processes", *Commun. Ass. Comput. Mach.*, vol. 22, May 1979.

[6] D. Dolev, M. Klawe and M. Rodeh, "An $O(n \log n)$ unidirectional algorithm for extrema-finding in a circle", *J. Algorithms*, vol. 3, Sept. 1982.

[7] W.R. Franklin, "On an improved agorithm for decentralized extrema finding in circular configurations of processors", *Commun. Ass. Comput. Mach.*, vol. 25, May 1982.

[8] G.N. Frederickson, "Tradeoffs for selection in distributed networks", in *Proc. 2nd ACM Symp. Principles of Distributed Computing*, Aug. 1983.

[9] R.G. Gallager, "Finding a leader in a network with $O(e) + O(n \log n)$ messages", Mass. Inst. Tech. Internal Memo., 1979.

[10] G.H. Gonnet, "A handbook of algorithms and data structures", U. Waterloo Tech. Rep. CS-80-23, 1980.

[11] D.S. Hirschberg, "Election processes in distributed systems", in *Proc. 18th Allerton Conf. Comm. Control and Comput.*, Oct. 1980.

[12] D.S. Hirschberg and J.B. Sinclair, "Decentralized extrema finding in circular configurations of processes", *Commun. Ass. Comput. Mach.*, vol. 23, Nov. 1980.

[13] E. Korach, S. Moran and S. Zaks, "Elections in complete graphs", in preparation.

[14] E. Korach, D. Rotem and N. Santoro, "Efficient distributed algorithms for finding centers and medians in a network", in *Proc. 18th Allerton Conf. Comm. Control and Comput.*, Oct. 1980.

[15] E. Korach, D. Rotem and N. Santoro, "A probabilistic algorithm for extrema-finding in a circle of processors", U. Waterloo Tech. Report CS 81-19, 1981.

[16] T. Minoura, "Ranking scheme and control token scheme", in *Proc. 2nd Symp. on Reliability in Distr. Software and Database Syst.*, July 1982.

[17] G. LeLann, "Distributed systems - toward a formal approach", in *Proc. IFIP*, 1977.

[18] J. Pachl, E. Korach and D.Rotem, "A Technique for proving lower-bounds for distributed election algorithms", *Proc. 14th ACM Symp. Theory of Comput.*, May 1982.

[19] G.L. Peterson, "An $O(n \log n)$ unidirectional algorithm for the circular extrema problem", *Trans. Progr. Lang. Syst.*, vol. 4, 198 .

[20] M. Rodeh, "Finding the median distributively", *J. Comput. Syst. Symp.*, vol. 24, 1982.

[21] D. Rotem, N. Santoro and J.B. Sidney, "A shout-echo algorithm for finding the median of a distributed set", in *Proc. 14th Southeastern Conf. Combin. Graph Theory and Comput.*, Feb. 1983.

[22] N. Santoro, "On the message complexity of distributed problems", Carleton U. Tech. Rep. SCS-TR-13, 1982.

[23] N. Santoro, E. Korach and D. Rotem, "Decentralized extrema finding in circular configurations of processors: an improved algorithm", in *Proc. 11th Conf. Num. Math. and Comput.*, Nov., 1981.

[24] N. Santoro, E. Korach and D. Rotem, "Decentralized rank finding in circular configurations of processors", U. Ottawa Tech. Rep. TR-81-4, 1981.

[25] N. Santoro and J.B. Sidney, "Order statistics on distributed sets", in *Proc. 20th Allerton Conf. Commun. Control and Comput.*, Oct. 1982.

[26] N. Santoro and J.B. Sidney, "Communication bounds for selection in distributed sets", Carleton U. Tech. Rep. SCS-TR- ,1982.

Appendix

Algorithm 3.1 below is described in terms of what operations a processor i must perform when in a given state upon certain conditions. Initially, all processors are in state 'available'.

Algorithm 3.1.

```
(1) In state 'available'.
    Upon reception of a locally originated request or of an 'A' message
      begin
        $\mathcal{L}_i := L_i; \bar{v}_i := i$;
        send ('A') to $L_i$;
        enter state 'active';
      end
(2) In state 'active'.
    If $i$ is a leaf: (let $x$ be the only neighbour)
      begin
        $v_i(x) := \delta$;
        send ('S', $i$) to $x$;
        enter state 'processing';
      end
If $i$ is not a leaf, upon reception of ('S', value) from $x \in L_i$:
      begin
        $\mathcal{L}_i := L_i - \{x\}$; $v_i(x) :=$ value;
        if $|\mathcal{L}_i| = 1$ then (an $S$ message has been received from all neighbours
        but one, say, $z$)
          $m := Min(\{v_i(y): y \in L_i - \{z\}\} \cup \{i\})$
          send ('S', $m$) to $z$ ;
          $v_i(z) := \delta$;
          enter state 'processing'
        endif
      end
(3) In state 'processing'
    Upon reception of ('S', value) from $x \in L_i$ :
      begin
        enter state 'saturated';
      end
    Upon reception of ('R', rank, min, sec) from $x \in L_i$ :
      begin
        $v_i(x) :=$ sec;
        enter state 'ranking';
      end
(4) In state 'saturated;
      begin
        if value > $m$ then
          $v_i(x) :=$ value;
```

```
            rank := 1 ;
            enter state 'ranking':
        else
            enter state 'processing';
        endif
    end
(5) In state 'ranking'.
    begin
        $m := Min(\{v_i(y) := y \in L_i\} \cup \{i\})$ ;
        if $m = i$ then
            my rank := rank;
            rank := rank + 1
            $\bar{v}_i := \infty$;
            $m := Min(\{v_i(y) : y \in L_i\})$;;
    end if
    if $m \neq \infty$ then
        $des := y \in L_i : v_i(y) = m$;
        $s := Min(\{v_i(y) : y \in L_i - \{des\}\} \cup \{i\})$;
        send ('R', rank, $m$, $s$) to des ;
        $v_i(des) := \delta$;
    endif
    if $m = \infty$ or $s = \infty$ then
        enter state 'terminating'
    else
            enter state 'processing'
    endif
    end
(6) In state 'terminating'.
    begin
        terminate the execution of the algorithm;
    end
```

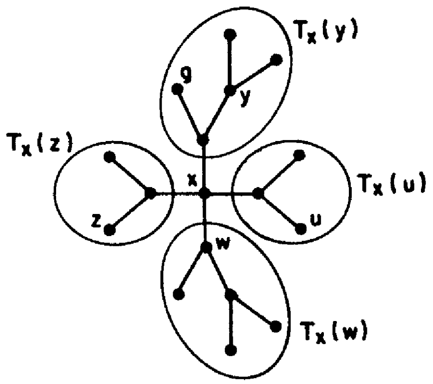

Fig. 1

Removing x and its incident edges we get a forest of four subtrees. Note that $T_x(y) = T_x(g)$.

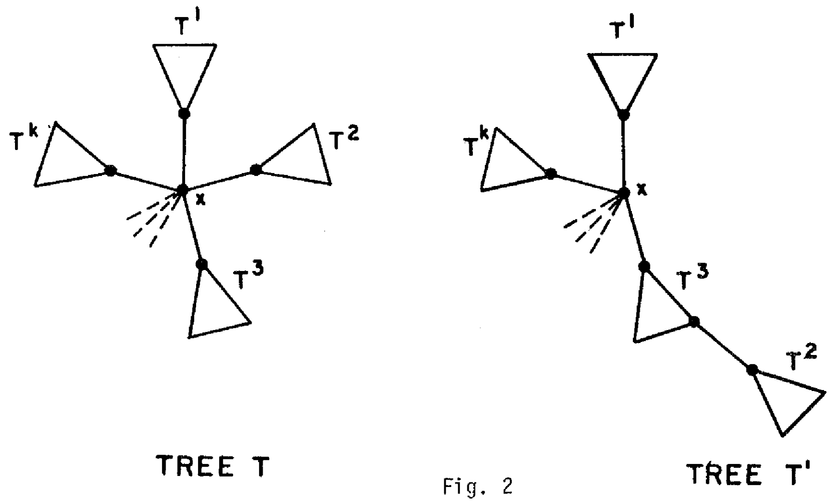

Fig. 2

The triangles denote the subtrees T^1, T^2 ... T^k obtained by removing x and its incident edges from T.

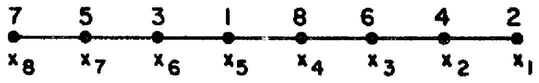

Fig. 3(a)

A worst case construction for n=8 requiring 31 messages.

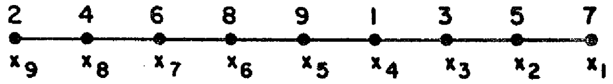

Fig. 3(b)

A worst case construction for n=9 requiring 39 messages.