Distributed Algorithms for Finding Centers and Medians in Networks

by

E. Korach (+)
D. Rotem (++)
N. Santoro (+++)

++Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-80-44
September 1980

Faculty of Mathematics

University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Distributed Algorithms for Finding Centers and Medians in Networks

by

E. Korach (+)
D. Rotem (++)
N. Santoro (+++)

++Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-80-44
September 1980

(+) Department of Combinatories and Optimization. University of Waterloo, Waterloo, Ontario, CANADA N2L 3G1.

(+++) Department of Computer Science. University of Ottawa, Ottawa, Ontario, CANADA K1N 9B4.

DISTRIBUTED ALGORITHMS FOR FINDING CENTERS AND MEDIANS IN NETWORKS

E. Korach [+]

D. Rotem [++]

N. Santoro [+++]

ABSTRACT

In this paper, we consider the problem of determining in a distributed fashion the centers and the medians of a network. Lower bounds on the time needed to solve these problems are proved. Algorithms that achieve those bounds for tree networks are presented; the number of exchanged messages is linear in the number of nodes. We extend these techniques to work on general networks in $O(n)$ time units exchanging $O(n \cdot e)$ messages, where n is the number of nodes and e is the number of edges in the network. In addition, a comparison with a simple heuristic approach is included.

Key Words and Phrases: Distributed algorithm, networks, center, median, analysis of algorithm.

* A preliminary version of this paper will appear in 18th Allerton annual conference.

DISTRIBUTED ALGORITHMS FOR FINDING CENTERS AND MEDIANS IN NETWORKS

E. Korach [+]

D. Rotem [++]

N. Santoro [+++]

ABSTRACT

In this paper, we consider the problem of determining in a distributed fashion the centers and the medians of a network. Lower bounds on the time needed to solve these problems are proved. Algorithms that achieve those bounds for tree networks are presented; the number of exchanged messages is linear in the number of nodes. We extend these techniques to work on general networks in $O(n)$ time units exchanging $O(n \cdot e)$ messages, where n is the number of nodes and e is the number of edges in the network. In addition, a comparison with a simple heuristic approach is included.

Key Words and Phrases: Distributed algorithm, networks, center, median, analysis of algorithm.

* A preliminary version of this paper will appear in 18th Allerton annual conference.

DISTRIBUTED ALGORITHMS FOR FINDING CENTERS AND MEDIANS IN NETWORKS

E. Korach, D. Rotem, N. Santoro

1. Introduction

Recently we are witnessing a wide and growing interest in the design and analysis of decentralized networks (for example MERIT or ARPANET) and distributed algorithms [1,2,5,6,8].

In distributed networks, topological information plays an important role. For example, in packet-switched store-and-forward networks, packets leaving a source are routed to intermediate nodes; thus, it is essential for every node in the network to have some knowledge of the topology of the network (eg. the adjacency matrix [14,15], the distance matrix [5], etc).

In general, information about the network topology can be usefully employed to develop efficient network algorithms; for example, the number of steps needed to synchronize the nodes can be minimized if a center of the network is known [10].

Unfortunately, topology information cannot be taken into account once and for all at design time. In fact, several unpredictable factors make the topology vary in time, eg. power failure, computer crash, link reactivation, etc. Therefore this information must be redetermined whenever it is needed.

In this paper we present and analyze distributed algorithms for locating centers and medians of a network. Such location algorithms are important in the design of routing mechanisms which provide broadcast with small delay [16].

The organization of the paper is as follows. In Section 2 the model is described. In Section 3, we consider the special important case of tree networks. A basic algorithm is presented; this algorithm activates all nodes in the network. An activated node, in turn, reports some information which is eventually collected at a certain node. It is shown that by processing the collected information we can design efficient algorithms to determine the center and the median of a tree.

Furthermore, we find lower bounds on the time required to find a center and a median in a tree and prove that our algorithms achieve these lower bounds and are therefore time optimal. The number of messages exchanged in both algorithms is shown to be linear in the number of nodes of the tree.

In Section 4 we extend the results of Section 3 to general networks. The idea here is to construct shortest path spanning trees for each node, and then locate the center or the median of the network by conducting a competition among the nodes of the network.

The exact solutions of our algorithms are compared with the approximate solutions obtained by heuristics that choose the center or the median of an arbitrary spanning tree. Achievable bounds on the accuracy of such approximate solutions show that they can be far from optimal. The proposed algorithms for finding a center and a median require at most $5 \cdot r(G)$ and $5 \cdot d(G)$ units of time respectively where $r(G)$ is the radius and $d(G)$ is the diameter of a network G. We also give upper bounds on the number of exchanged messages.

2. The Model

In this section we describe the framework and define some terms that are used throughout the paper.

A communication network can be represented as a graph $G = (V(G), E(G))$ where $V(G)$ is a set of nodes and $E(G) \subseteq V(G) \times V(G)$ is a set of arcs. An arc $(i,j) \in E(G)$ represents a bidirectional communication link between nodes i and j.

We denote by $INF(i,t)$ the information about the network stored at node i at time t. We assume that for $i \neq j$ $INF(i,0) \not\subseteq INF(j,0)$, i.e., before any exchange of information occurs, no node i knows the information of node j, for all $i,j \in V(G)$. Each node i makes a decision at time t which is a function of the total information known to i at this time. For each node i, $INF(i,0)$ includes a list of its neighbours $N(i)$ where

$$N(i) = \{j \mid (i,j) \in E(G)\}.$$

To illustrate the above assumptions, consider two nodes $k,l \in V(G)$ where $l \in N(k)$ and $N(l) = \{k\}$. Then k knows that $\{k\} \subseteq N(l)$ but does not know $\{k\} = N(l)$; hence even in this simple case

$$INF(l,0) \not\subseteq INF(k,0).$$

Unless otherwise stated, we will use the following standard assumptions in our algorithms:

1) Partial reliability -- the network topology does not change during the execution time of an algorithm.
2) The network is synchronous and it takes one unit of time to transmit a message along any edge.
3) The time to process a message at a node (including its queueing time) is negligible when compared with the time to transmit a message along an edge.

We now list the graph theoretical terms commonly used in the paper; the reader is referred to any standard graph theory text (for example [4]) for definitions of any additional terms.

Given a graph $G = ((V(G), E(G))$ we denote by:

$d(i,j)$ – distance between $i,j \in V(G)$;

$r(i) = \max_{j \in V(G)}(d(i,j))$ – maximum distance between i and any other node;

$c(G) = \{j \mid r(j) = \min_{i \in V(G)} r(i)\}$ – set of centers of G;

$DIS(G,i) = \sum_{j \in V(G)} d(i,j)$ – sum of distances from a node i;

$m(G) = \{j \mid DIS(G,j) = \min_{i \in V(G)}(DIS(G,i))\}$ – set of medians of G;

$r(G) = r(i)$ where $i \in c(G)$ – radius of G;

$d(G) = \max_{i \in V(G)} r(i)$ – diameter of G.

In a tree T we denote by:

$\langle x,y \rangle$ – the unique path between $x,y \in V(T)$;

diameter path – a longest path in T, i.e. any path with length $d(T)$;

$h(T)$ – the height of T – (defined only for rooted trees) the length of the longest path from the root to any leaf.

3. Tree Networks

3.1 A Basic Algorithm

In this section we present a basic algorithm which is a building block in the construction of the distributed algorithms presented in this paper.

The purpose of this algorithm is to collect some relevant information from all nodes in the network and make it available to one or more nodes called SATURATED nodes. The SATURATED nodes can subsequently use this information to locate nodes with required properties (e.g. centers, medians, nodes with highest identifiers, etc.) or to determine properties of the network (e.g. diameter, number of nodes, etc.).

In the following sections we will use $S(t)$ to denote the set of nodes activated after t units of time from the start of the algorithm.

An arbitrary node I, called the initiator, activates all nodes in the network as follows:

(a) The initiator I sends a FORWARD message to all its neighbours.

(b) When a node is activated by a FORWARD message from one of its neighbours it sends a FORWARD message to all its other neighbours.

When a leaf (a node of degree one) is activated, it sends a BACKWARD message containing some information. The nature of this information may vary according to the context in which the basic algorithm is applied. For all other activated nodes which are not leaves, the algorithm proceeds as follows. At each time instance t, each node i checks which BACKWARD messages were received so far; it will take some actions only on the following cases:

(a) If all but exactly one neighbour, say j, have already sent BACKWARD messages to i by this time, then i sends a BACKWARD message to j.

(b) If BACKWARD messages were received from all neighbours then i becomes SATURATED. □

In the very special case when the initiator I is a leaf, then I sends the BACKWARD message to its only neighbour one time instance after the FORWARD message has been sent. In Section 3.1.2, a formal description of this algorithm is given.

3.1.1 Basic Properties

We need some terminology in order to prove interesting properties of the basic algorithm. In a tree T, every node $x \in V(T)$ with $|N(x)| = k$ is the root of k subtrees $\tau = \{T_1, \ldots, T_k\}$ such that for $i \neq j$, $T_i \cap T_j = \{x\}$, and $\bigcup_{i=1}^{k} T_i = T$. For a node $l \neq x$ we denote by T_{lx} the unique tree in τ which contains l.

Let $D = \langle a,b \rangle$ be a diameter path in a tree T.

Lemma 3.1:

A node $x \in D$ can send a BACKWARD message only to a neighbour on D.

Proof:

If $x = I$ then the lemma trivially holds. Let $x \neq I$ be a node on D with $N(x) = \{1, 2, \ldots, k\}$. After node x changes its state to FORWARD, it sends messages to every $i \in N(x)$ such that $i \notin T_{Ix}$. Let us denote by $t_x(i)$ the time it takes for a report from $i \in N(x)$ to arrive at x, measured from the time that x becomes FORWARD. For T_{ix} that does not contain I, $t_x(i)$ is the time needed for the message to traverse the distance from x to the furthermost node in T_{ix} and then back to x. The only exception is T_{Ix} on which a FORWARD message was sent prior to the time that x became FORWARD. Hence

$$t_x(i) = 2h(T_{ix}) \quad \text{for } i \notin T_{Ix}$$

$$\text{and} \tag{3.1}$$

$$t_x(i) < 2h(T_{Ix}) = 2h(T_{ix}) \quad \text{for } i \in T_{Ix}$$

If x becomes SATURATED then it does not send any BACKWARD message, and the lemma trivially holds. Otherwise let $l \in N(x)$ be the unique last node which has not reported yet to x at the time that x becomes BACKWARD. This node will receive the BACKWARD message from x. We now show that $l \in D$. By eq. (3.1)

$$\text{either} \quad h(T_{lx}) > h(T_{ix}), \quad \text{where } T_{ix} \neq T_{lx} \tag{3.2}$$

$$\text{or} \quad h(T_{Ix}) \geqslant h(T_{lx}) > h(T_{ix}) \quad \text{such that } T_{ix} \neq T_{Ix}, \; T_{ix} \neq T_{lx}$$

in both cases $l \in D$ and the lemma is proved. □

Lemma 3.2

For any diameter path D, a node $y \notin D$ cannot be SATURATED.

Proof:

Let x be the nearest node to y on D and let $<x = x_0, x_1, x_2, \ldots, x_m = y>$ be the path from x to y. By Lemma 3.1 since $x_i \notin D$ for $1 \leqslant i \leqslant m$, x_1 will not receive a BACKWARD message from x, and in general x_{i+1} does not receive a message from x_i, $1 \leqslant i \leqslant m-1$. Finally y does not receive a BACKWARD message from x_{m-1}, hence y cannot be SATURATED.□

Theorem 3.1

Either one or two nodes become SATURATED in finite time, and every SATURATED node lies on all diameter paths.

Proof:

Every node must change its state to FORWARD after at most $d(T)$ units of time and then changes its state to BACKWARD after at most an additional $2 \cdot d(T)$ units of time. Therefore by $3 \cdot d(T)$ time units, every node either becomes SATURATED or BACKWARD.

Let us choose an arbitrary node s and traverse a BACKWARD edge going out of s, i.e. an edge along which a BACKWARD message was sent. By traversing BACKWARD edges in this way we must meet a SATURATED node since there are no cycles in T. Hence there exists at least one SATURATED node s. Assume that there are more than two SATURATED nodes, then there exist two SATURATED nodes s_1 and s_2 such that $d(s_1, s_2) \geqslant 2$. Consider any node x on the path $<s_1, s_2>$; x could not send a BACKWARD message in the direction of both s_1 and s_2 and therefore one of these nodes cannot be SATURATED. We conclude therefore that there are one or two SATURATED nodes and by Lemma 3.2 all such nodes must lie on all diameter paths. □

It is easy to show that the number of messages exchanged until a SATURATED node is found is $2(n-1)$ if there is a single SATURATED node and $2n-1$ otherwise. This follows because exactly two messages will be transmitted along each edge with the exception of the edge between two SATURATED nodes on which three messages will be exchanged. We defer the analysis of the number of time instances to the next section.

3.1.2 Formal Description of the Basic Algorithm

Each node i has available to it:

(i) the queue $Q(i)$ of current messages; each entry consists of a pair [message, sender].

(ii) the list $N(i)$ of its neighbours in the network.

(iii) a state indicator "status"; the possible values of "status" are: INACTIVE, FORWARD, BACKWARD and SATURATED.

The algorithm uses the functions DEQUEUE and ENQUEUE defined as follows:

DEQUEUE $[M, i]$

removes one entry from the queue Q (current node) and sets M and i to the value of the "message" and "sender" of the removed entry, respectively;

ENQUEUE $Q(j)[M$, current node]

enters the entry [M, current node] into the queue of node j, i.e. it sends [M, current node] to node j.

Initially each node is in state "INACTIVE" and $N_i := N(i)$ for all i.

An arbitrary node I starts the algorithm by performing the following operations:

```
INITIAL
begin
        status := "FORWARD";
        for all j ∈ N(I) ∪ I do
        begin
                dest := j;
                M := "message";
                SEND;
        end
end;
```

The algorithm uses the following routines:

```
SEND
begin
        ENQUEUE Q(dest)[M,i];
end;
```

```
SATURATED
begin
        status := "SATURATED";
end;
```

```
PROCESSMESSAGE
begin
        "perform needed operations"
end;
```

The BASIC algorithm will be as follows.

```
begin
    while Q(i) ≠ ∅ do
    begin
        DEQUEUE [M,k];
        if status = "INACTIVE" then
        begin
            if | N(i)| > 1 then
            begin
                status := "FORWARD";
                for all j ∈ N(i) − k do
                begin
                    dest := j;
                    SEND;
                end
            end
            else
            begin
                status := "BACKWARD";
                dest := k;
                SEND;
            end
        end
        else
        begin
            if status = "FORWARD" then
            begin
                N_i := N_i − k;
                if N_i = ∅ then SATURATED
                else
                begin
                    PROCESSMESSAGE;
                    if (| N_i| = 1 and Q(i) = ∅) then
                    begin
                        status := "BACKWARD";
                        dest := N_i;
                        SEND;
                    end
                end
            end
        end
    end
end;
```

3.2 Finding a Center in a Tree

As mentioned before, the knowledge of a center and of the radius of a network can be usefully employed in several algorithms to either minimize or bound the maximum transmission time and the volume of exchanged messages [16].

3.2.1 Basic Properties

We present here some basic properties of trees which are used in our algorithm.

Given a tree T:

Property 3.1:

If $d(T)$ is even there is a unique center, else there are two centers at distance one from each other.

Property 3.2:

$$r(T) = \lceil \frac{d(T)}{2} \rceil .$$

Property 3.3:

Every diameter path contains all centers of T.

Proof:

Given any diameter path $D = <a,b>$ and a node $x \notin D$, x cannot be a center because clearly

$$\max\{d(x,a),\ d(x,b)\} > r(T).$$

Hence every center must lie on all diameter paths. □

Property 3.4:

For a node $x \in T$, let j be the furthermost node from x in T; i.e., $d(x,j) = r(x)$. Then

$$d(j,c) \geqslant \lfloor \frac{d(T)}{2} \rfloor$$

where c is a center of T.

Proof:

By contradiction. Let $D = <a,b>$ be a diameter path, then either $c \in <x,b>$ or $c \in <x,a>$. Assume $c \in <x,b>$, then

$$d(x,b) = d(x,c) + d(c,b) \geqslant d(x,c) + \lfloor \frac{d(T)}{2} \rfloor . \tag{3.3}$$

On the other hand, by the triangle inequality and by the assumption $d(c,j) < \lfloor \frac{d(T)}{2} \rfloor$,

$$d(x,j) \leqslant d(x,c) + d(c,j) < d(x,c) + \lfloor \frac{d(T)}{2} \rfloor ; \tag{3.4}$$

hence, $d(x,j) < d(x,b)$ which contradicts the fact that j is the furthermost node from x. □

3.2.2 The Algorithm TREE-CENTER

A distributed algorithm for finding the center in a tree was presented in [12]. To make the present paper self contained, we present a formal version of that algorithm. Furthermore, we will prove here that the algorithm is time optimal.

The algorithm is based on the observation that if a node x on the diameter path knows its distance from the two extreme points of this path, it can determine both the direction and the distance from itself to the closest center. This observation is formally stated in the following lemma:

Lemma 3.3:

Let x be on a diameter path $D = <a,b>$, $d(x,a) = p_1$, and $d(x,b) = p_2$; then

(a) if $p_1 \geqslant p_2$ then c lies on $<x,a>$, otherwise it lies on $<x,b>$; and

(b) $d(x,c) = \lfloor \frac{|p_1 - p_2|}{2} \rfloor$

Proof:

Simple use of properties 3.1, 3.2 and 3.3. □

In order to find a center, we use the BASIC algorithm as described in Section 3.1.2. We use the BACKWARD messages to determine maximum distances from the SATURATED node. To achieve this, the leaves send BACKWARD messages with counters set to 1. These counters are incremented by one at each traversed node. The two largest counters that reach a SATURATED node s contain the distances from s to the two extremes of a diameter path $D = <a,b>$, (see *Theorem 3.1*). By Lemma 3.3 a SATURATED node knows $d(s,c)$ and whether $c \in <s,a>$ or $c \in <s,b>$.

The algorithm TREE-CENTER will be the BASIC algorithm where in INITIAL (see Section 3.1.2) the "message" is set to zero and the operations SEND, SATURATED and PROCESSMESSAGE are specified below. Let us observe that using this algorithm all nodes between the first SATURATED node(s) and the center will become SATURATED.

```
SEND
begin
      if status = "BACKWARD" then M:=M+1
      ENQUEUE Q(dest)[M,i];
end
```

The procedure SATURATED is based on the observation that maxmessage and M (the last message received) are the two largest counters received by a SATURATED node.

```
SATURATED
begin
      status := "SATURATED";
      Diff := M - maxmessage;
      if |Diff| ≤ 1 then status := "CENTER"
      else
      begin
            if Diff ≥ 2 then
            begin
                  inf := maxmessage + 1;
                  dest := k;
            end;
            else
            begin
                  inf := M + 1;
                  dest := maxsender;
            end;
            ENQUEUE Q(dest)[inf,i];
      end;
end;
```

```
PROCESSMESSAGE
begin
      if M > maxsender then
      begin
            maxsender := k;
            maxmessage := M;
      end;
end;
```

3.2.3 Analysis of the Algorithm

In this section we prove correctness and optimality of algorithm TREE-CENTER.

Lemma 3.4

Algorithm TREE-CENTER correctly finds a center in a tree T.

Proof:

The correctness of this algorithm is based on the following observation. Let x be in the FORWARD state, and assume it receives the message M from its neighbour y. Then $M = h(T_{yx})$; i.e., M is the distance between x and the furthermost node in T_{yx}. This fact is easily proved by induction on the height of T_{yx}. It therefore follows that when a node s becomes SATURATED it knows its distance from the furthermost nodes in T_{is} for all $i \in N(s)$. Since s is on a diameter path (*Theorem 3.1*), the two largest messages p_1 and p_2, received at s, specify the distances from s to the two extreme points on a diameter path on which it lies, and by *Lemma 3.1* s sends a message to the correct direction of the center. Since $|p_1 - p_2|$ decreases by 2 at each node traversed on the path from s to the center, eventually a SATURATED node for which $|p_1 - p_2| \leqslant 1$ must exist and this node changes its state to CENTER.□

In order to prove a lower bound on the time needed to find a center in a tree we use the equation

$$\mathrm{INF}(i,t) = \mathrm{INF}(i,0) \quad \text{if } i \notin S(t)$$

$$\text{or} \tag{3.5}$$

$$\mathrm{INF}(i,t) \subseteq \mathrm{INF}(i,t-1) \cup \{\mathrm{INF}(j,t-1) \mid j \in N(i) \cap S(t-1)\}.$$

where $S(t)$ is the set of nodes which were activated by time t.

In words, the information known to node i at time t is $\mathrm{INF}(i,0)$ if i has not been activated yet, else it is a subset of the information known at time $t-1$ to itself and its activated neighbours.

Theorem 3.2

Any distributed algorithm for finding a center in a tree T from originator X requires at least $B(X) = r(X) + \lfloor \frac{d(T)}{2} \rfloor$ time units.

Proof:

Assume that an algorithm A finds that c is a center of T at time $t_f < B(X)$. Then the maximum information at node c at this time is

$$\mathrm{INF}(c,t_f) \subseteq \mathrm{INF}(c,t_f-1) \cup \{\mathrm{INF}(j,t_f-1) \mid j \in N(c) \cap S(t_f-1)\} \tag{3.6}$$

by eq. (3.5). By repeated application of eq. (3.6) we get

$$\mathrm{INF}(c,t_f) \subseteq \{\mathrm{INF}(j,0) \mid j \in S(t_f - d(j,c))\}, \tag{3.7}$$

i.e., the information at c at time t_f includes information from node j, if and only if j was active at time $t_f - d(j,c)$.

Let j be a node in T such that $d(j,X) = r(X)$. We will show that the information from j cannot arrive at c by time t_f.

First we note that

$$j \notin S(r(X) - 1) \tag{3.8}$$

therefore

$$j \notin S(t_f - \lfloor \frac{d(T)}{2} \rfloor) \tag{3.9}$$

since by our assumption on t_f

$$t_f - \lfloor \frac{d(T)}{2} \rfloor \leqslant r(X) - 1.$$

By Property 3.4. $d(j,c) \geqslant \lfloor \frac{d(T)}{2} \rfloor$ and hence we have $j \notin S(t_f - d(j,c))$ which implies $\mathrm{INF}(j,0) \subsetneq \mathrm{INF}(c,t_f)$ by (3.7). Next, we modify the tree T by adding a path $<j,j_1,j_2>$ rooted at j, (see Figure 1) and call the obtained tree T'. We show that when A operates on T' it makes a wrong decision.

At time t_f, $\mathrm{INF}(c,t_f)$, when A is applied to T', is the same as $\mathrm{INF}(c,t_f)$ when A is applied to T. This follows because by the above arguments no information from j_1 or j_2 can reach c in T', and all other nodes are activated at the same time both in T and T'. Hence at time t_f, A will choose c as the center of T' which is clearly a wrong decision.□

Theorem 3.3:
Algorithm TREE-CENTER is time optimal.

Proof:

Let x be the furthermost node from I in T. By Property 3.4 we know that x is an extreme of a diameter path. Let c be the center closest to x (see Fig.2), i.e. $d(x,c) = \lfloor \frac{d(T)}{2} \rfloor$.

Let us consider the situation after $F = d(I,x) + \lfloor \frac{d(T)}{2} \rfloor - 1$ time units from the beginning of the algorithm. Two cases are theoretically possible.

(a) No node in T is SATURATED: In this case in time $F + 1$, c will be SATURATED. This follows since the node x was activated and its BACKWARD message reaches c by time $F + 1$, also the BACKWARD messages from all other nodes in T have already reached c by this time. Hence c must be SATURATED and the algorithm correctly locates the center in optimal time.

(b) A node s became SATURATED by time F: Let us consider the SATURATED node s closest to c. By Theorem 3.1, s lies on all diameter paths. Furthermore, $s \in <x,c>$ since if $s \notin <x,c>$ the BACKWARD message from x cannot arrive at s by time F, contradicting the fact that that s is SATURATED by this time. Therefore s becomes SATURATED at time

$$d(x,I) + d(s,x).$$

After additional $d(s,c)$ time units the center c becomes SATURATED and therefore the algorithm is optimal also in this case and terminates at time $d(x,I) + d(x,s) + d(s,c) = d(x,I) + d(x,c)$. □

It is easy to show that the number of messages exchanged by algorithm TREE-CENTER is:

$$\begin{cases} 2(n-1)+r^* & \text{if there is a unique SATURATED node} \\ 2(n-1)+r^*+1 & \text{if there are two SATURATED nodes} \end{cases}$$

where r^* is the minimum distance between a SATURATED node and a center. This follows because on each edge, two messages are sent (FORWARD and BACKWARD) whereas on every edge of $<s,c>$ a SATURATED message is added.

3.3 Finding a Median in a Tree

Another important node in a tree is a median, i.e. a node from which the average distance to all nodes in the tree is minimized [16]. In this section we show how to apply the BASIC algorithm in order to find the median.

3.3.1 Basic Properties

Lemma 3.5:

Let $x,y \in V(T)$ be connected by an edge then

$$\mathrm{DIS}(T,x) = \mathrm{DIS}(T,y) + |V(T_{xy})| - |V(T_{yx})|. \tag{3.10}$$

Proof:

$$\mathrm{DIS}(T,x) = \sum_{z \in T} d(x,z) = \sum_{z \in V(T_{yx}) - \{x\}} d(x,z) + \sum_{z \in V(T_{xy}) - \{y\}} d(x,z), \tag{3.11}$$

since $d(x,x) = 0$ and $[V(T_{yx})] \cup [V(T_{xy}) - \{y\}] = V(T)$. By

$$d(x,z) = \begin{cases} d(y,z)+1 & \text{for } z \in V(T_{yx}) - \{x\} \\ d(y,z)-1 & \text{for } z \in V(T_{xy}) - \{y\} \end{cases}$$

we have

$$DIS(T,x) = \sum_{z \in V(T_{xy}) - \{x\}} d(y,z) + (|V(T_{xy})| - 1) + \sum_{z \in T_{yx} - \{y\}} d(y,z) - (|V(T_{yx})| - 1).$$

from which (3.10) follows. □

Let $\Delta(x,y) = \mathrm{DIS}(T,x) - \mathrm{DIS}(T,y)$ and $|V(T)| = N$.

Corollary 3.1:

$$\Delta(x,y) = N + 2 - 2|V(T_{xy})| = -\Delta(y,x).$$

Proof:

Follows from (3.10) and the fact that

$$|V(T_{xy})| = N + 2 - |V(T_{yx})| \; \square$$

Lemma 3.6:

For $x \in T$ let $N(x) = \{1, 2, \ldots, j\}$ be the set of neighbours of x in T. There exists at most one vertex $i \in N(x)$ such that

$$\Delta(i,x) \leqslant 0.$$

Proof:

We derive a contradiction by assuming that for $k \neq l$, $k,l \in N(x)$ both $\Delta(x,k)$ and $\Delta(x,l)$ are non positive.

$$\Delta(k,x) = N + 2 - 2|V(T_{kx})| \leqslant 0 \tag{3.12}$$

$$\Delta(l,x) = N + 2 - 2|V(T_{lx})| \leqslant 0. \tag{3.13}$$

By adding (3.12) and (3.13) we get

$$2(N + 2) - 2(|V(T_{kx})| + |V(T_{lx})|) \leqslant 0.$$

But this is impossible since

$$|(V(T_{kx})| + |V(T_{lx})| \leqslant N + 1. \; \square \tag{3.14}$$

Lemma 3.7

A node $x \in V(T)$ is a median of T if and only if for all $i \in N(x)$,

$$\Delta(i,x) \geqslant 0.$$

Proof:

"Only if" - let x be a median then DIS $(T,i) \geqslant$ DIS(T,x) for all $i \in V(T)$, hence for $i \in N(x)$, $\Delta(i,x) =$ DIS$(T,i) -$ DIS$(T,x) \geqslant 0$.

"if" - let m be a median of T and let us choose a node y which is not a median. We now show that y has a neighbour i for which $\Delta(i,y) < 0$, i.e. DIS$(T,y) >$ DIS(T,i). Let $<y = y_1, y_2, \ldots, y_l, m>$ be the path from y to m in T; if $y_l = y_1$ then $\Delta(y_l,m) > 0$ (since y_l is not a median) and by Corollary 3.1 $\Delta(m,y_l) < 0$, and the lemma is proved. Otherwise $\Delta(y_l,m) \geqslant 0$ (by minimality of DIS(T,m)), and by Corollary 3.1 $\Delta(m,y_l) \leqslant 0$; hence by Lemma 3.6, $\Delta(y_{l-1},y_l) > 0$. By repeating this argument along the path $<y,m>$ we get $\Delta(y_2,y) = \Delta(y_2,y_1) < 0$ which completes the proof of the lemma.□

Corollary 3.2

In a tree T with median m, for a node $y \in V(T)$ which is not a median, there exists a unique $i \in N(y)$ with $\Delta(i,y) < 0$. In this case $i \in <y,m>$ and

$$| V(T_{iy})| > | V(T_{jy})| \quad \text{for } j \in N(y) - \{i\}.$$

Proof:

Simple use of Lemmas 3.6 and 3.7.□

Corollary 3.3:

There are at most two medians in a tree.

Proof:

Let m be a median, then by Lemma 3.6, m may have at most one neighbour, say m', which is a median, i.e. $\Delta(m,m') = 0$. Also if any other median m'' exists, then by following the arguments from Lemma 3.7 along the path $<m'',m>$ we derive a contradiction.□

From all the above properties we can see that if a SATURATED node s knows all $\Delta(i,s)$ for $i \in N(s)$ it can determine whether s itself is a median (all $\Delta(i,s) \geqslant 0$) or if there exists $j \in N(s)$ such that $\Delta_{js} < 0$. In the latter case the median lies in the direction of j and s reports to j which will become SATURATED.

3.3.2 The Algorithm TREE-MEDIAN

By the properties proved in the previous section, it is clear that if a node x knows DIS(T_{ix},x) and $| V(T_{ix})|$ for all $i \in N(x)$ it can compute $\Delta(i,x)$ and decide whether x is itself a median and, if not so, it can determine the direction of the median.

The purpose of our algorithm is to supply this information to the first SATURATED node from which the median is approached by a sequence of SATURATED nodes. This is done by providing two counters, m_1 and m_2, with each BACKWARD message sent. When a node x sends a BACKWARD message to y then

$$m_1 = DIS(T_{xy},x) - 1 \quad \text{and} \quad m_2 = | V(T_{xy})| - 1.$$

Node y can process all BACKWARD messages received such that, when y becomes BACKWARD, it knows $DIS(T_{iy},y)$ and $| V(T_{iy})|$ for all but one neighbour to which it sends a BACKWARD message. This process continues until some node becomes SATURATED and the median can be found. The algorithm TREE-MEDIAN is the BASIC algorithm where the operations SEND, PROCESSMESSAGE and SATURATED are described below.

```
PROCESSMESSAGE
begin
        countdist := m1 + m2 + countdist;
        countnode := countnode + m2;
        if m2 > maxnode then
        begin
                maxnode := m2;
                maxdist := m1;
                maxsend := k;
        end
end;


SEND
begin
        if status = "BACKWARD" then
        begin
                m1 := countdist;
                m2 := countnode + 1;
                M := (m1,m2)
        end
        Enqueue Q(dest)[M,i];
end;
SATURATED
begin
        status := "SATURATED";
        PROCESSMESSAGE;
        DELTA := (countnode + 1 + 2) - (2 · maxnode);
        if DELTA < 0 then
        begin
                m1 := countdist - (maxdist + maxnode)
                m2 := countnode - maxnode + 1;
                dest := maxsend;
                SEND;
        end
        else status := "MEDIAN";
end;
```

We now state a result analoguous to the one proved for finding the center in a tree.

Theorem 3.4:

Any distributed algorithm for finding the median of a tree T from initiator I, requires at least

$$\max_{x \in T}(d(I,x) + d(x,m))$$

units of time.

Theorem 3.5:

Algorithm TREE-MEDIAN is time optimal for all trees with $|V(T)| \geqslant 3$.

The proofs of both theorems follow the same lines as those of theorems 3.1 and 3.2, and are therefore omitted here.

4. General networks

4.1 Basic Algorithms

In many cases, efficient distributed algorithms for finding certain properties of general networks, can be found by constructing certain spanning trees and then applying known algorithms which operate on trees [5,10,11]. Using this principle, we combine here the results of the previous section with a known algorithm for constructing spanning trees in order to find the center and the median of a general network.

Let us briefly describe a well known algorithm [9] that generates a spanning tree $SPT(x)$ rooted at a node $x \in V(G)$, such that the distance from x to any node $v \in V(G)$ in $SPT(x)$ is equal to $d(x,v)$ in G.

Algorithm SPANNING-TREE

Step 1.

x sends a message SP to all $i \in N(x)$.

Step 2.

When a node v receives the first SP message (ties are broken arbitrarily), it sends an ACKNOWLEDGE message along this edge and an SP message along all other edges incident with v.

Step 3:

Every node v marks all the neighbours in G from which it received and to which it sent an ACKNOWLEDGE message. These are the neighbours of v in $SPT(x)$.

4.2 Finding Centers of General Networks

Our algorithm is based on some relations between the centers of a network and the centers of $SPT(x)$ for $x \in V(G)$. In general the radius of $SPT(x)$ for an arbitrary $x \in V(G)$ can be as large as $d(G)$ as shown in the next example.

Example:

Consider the $n \times n$ grid graph G in Figure 3. The tree $SPT(I)$ is shown in bold lines. $I \in c(SPT(I))$ and $r(SPT(I)) = 2(n-1)$ where $r(G) = n-1$.□

Therefore the simple heuristic of choosing a center in $c(SPT(x))$ of an arbitrary spanning tree $SPT(x)$ as an approximation, may be as bad as choosing an arbitrary node. However we know that a center of G is also a center of its own spanning tree as proved in the following lemma.

Lemma 4.1:

If $x \in c(G)$ then $x \in c(SPT(x))$.

Proof:

For all $y \in V(G)$ we have

$$h(SPT(y)) \geq r(G) = h(SPT(x)) = r(x)$$

hence x must be a center of $SPT(x)$.□

From this simple observation it follows that the only candidates of being centers of the graph are those nodes which are centers of their own spanning trees. We are now in a position to describe the algorithm GRAPH-CENTER starting from initiator I.

Algorithm GRAPH-CENTER

(a) Create a spanning tree $SPT(I)$

(b) Whenever a node x is activated in (a), it will initiate a process of finding the center of its own spanning tree $SPT(x)$.

(c) If node x finds that it is the center of $SPT(x)$ it is a candidate for being a center of G, it will then report $r(SPT(x))$ to I along the path $<x,I>$ in $SPT(I)$. This requires that in (a), each node marks its activator in $SPT(I)$.

(d) The initiator I will choose a center of G to be the node x which sent the minimum $r(SPT(x))$ within time $W = 4r(G) + 1$, and then will notify it.□

We now prove the correctness of the algorithm and show how the initiator I can compute W.

Lemma 4.2:

A message from a center x of G will arrive at I after t_x time units where $t_x \leqslant 4 \cdot r(G) + 1$.

Proof:

The message which activates x in (a) will reach x in $d(I,x)$ time units. By Lemma 4.1, x is the center of $SPT(x)$, therefore after at most additional $2 \cdot h(sPT(x)) + 1 = 2 \cdot r(x) + 1$ time units (see Theorem 3.3) x will find that it is a center of $SPT(x)$. The report sent by x back to I will require an additional $d(I,x)$ time units.

Summarizing,

$$t_x = 2d(I,x) + 2r(x) + 1 \leqslant 4r(G) + 1.$$

This completes the proof.□

This lemma guarantees that after $4 \cdot r(G) + 1$ time units, the initiator must have received a message from a center of G.

We now show how the initiator I can compute W without knowing $r(G)$. Initially I sets $W := \infty$, and its internal clock $t := 0$. Every time a report from a node x containing $r(SPT(x))$ is received, I updates its information about the minimum radius and the node that achieves this minimum:

```
if 4·r(SPT(x)) + 1 < W then
begin
        W := 4·r(SPT(x)) + 1;
        tempc := x;
end;
```

and at every time instance, I performs the following operations:

```
t := t + 1;
if t = W then tempc is a center.
```

Summarizing, the entire algorithm GRAPH-CENTER requires in total $W + d(I,c)$ time units where c is the closest center to I. This is bounded by $4 \cdot r(G) + 1 \leqslant W + d(I,c) \leqslant 5 \cdot r(G) + 1$.

An example of a graph in which our algorithm attains its upper bound is given in Figure 4. The lower bound will be achieved on any graph in which I is a center.

The number of messages exchanged in this algorithm is fairly large. Each node has to participate in the construction and 'center finding' of n spanning trees. In each such spanning tree a node x will send at most $\deg(x) + 1$ messages to find the center, where $deg(x)$ is the degree of x in G. In total this requires

$$\sum_{x \in V(G)} (deg(x) + 1) = 2|E(G)| + |V(G)| \quad \text{messages.}$$

In the worst case every node has to report to I and this may require $DIS(SPT(I),I)$ additional messages. Finally, I has to notify the center; this requires at most $r(G)$ messages.

Since $DIS(SPT(I),I) \leqslant n^2$ we have an upper bound of

$$2n|E(G)| + 2n^2 + r(G)$$

on the number of exchanged messages where $n = |V(G)|$.

4.3 Finding the Median in a Graph

An algorithm for finding a median can be devised along the same lines as the GRAPH CENTER algorithm, where now each node x reports to I the value of $DIS(SPT(x),x)$ (the sum of distances in $SPT(x)$ from x to any other node). Unlike GRAPH-CENTER, the initiator I has to receive messages from all nodes in order to determine a median.

Algorithm GRAPH-MEDIAN

(a) The initiator I creates a spanning tree $SPT(I)$

(b) Whenever a node x is activated in (a) it will find $DIS(SPT(x),x)$ in $SPT(x)$ and report it to I.

(c) Initiator I chooses x, whose report is minimum, as the median, and then notifies x.□

In the worst case this algorithm requires 5d(G) units of time. It can also be shown that at most $2n|E(G)| + n^2 + d(G)$ messages will be exchanged.

The large number of messages exchanged by algorithm GRAPH-MEDIAN might motivate one to find approximate solutions.

In general, the simple heuristic of choosing the median m_x of an arbitrary tree $SPT(x)$ for some $x \in V(G)$, as an approximation to the correct median m of G, can produce undesirable results. In fact, the ratio between $DIS(G,m_x)$ and $DIS(G,m)$ can be unbounded as shown in the following example.

Example:

Consider the family of graphs of Figure 5(a). It is possible that the tree $SPT(I)$ will have the shape as shown in figure 5(b). In this case $m_I = I$ and $DIS(G,I) = k^2 + (2l + 1)k + 2$ while $DIS(G,m) = k^2 + k + 2l$. Clearly when $l = k^\alpha$ $\alpha \geqslant 2$, $\lim_{k \to \infty} \frac{DIS(G,I)}{DIS(G,m)} = \infty$. □

5. Concluding Remarks

Centers and medians play an important role in the construction of optimal broadcasting schemes and synchronization techniques in decentralized networks.

In this paper we have considered the problem of devising efficient distributed algorithms for finding a center and a median in a network. These algorithms can easily be extended to find all centers and all medians in a network.

The main contributions of this work are:

(a) Presenting a basic algorithm which can be applied for finding different properties of networks.

(b) Proving lower bounds on the time required to find centers and medians, in a tree network.

(c) Presenting time optimal algorithms for tree networks; the number of exchanged messages is linear in the number of nodes.

(d) Presenting algorithms for finding medians and centers for general networks. For a network with n nodes and e edges both algorithms run in $O(n)$ time units and require $O(ne + n^2)$ exchanged messages.

Throughout this paper, we have considered the case of only one node starting the process of finding the center or median of the network. This apparent limitation does not affect the generality of our algorithms. In fact, the case of several initiators can be reduced to the case of a simple initiator by means of a collision resolution mechanism based on a simple priority function [10].

In [7], lower bounds on the time needed to find centers medians and other important properties of general networks are presented. There seems to be a large gap between these lower bounds and the upper bounds presented here. Further research is needed to reduce this gap and also to improve the upper bounds on the number of exchanged messages.

It seems that in a general network, finding the exact center and median is a costly operation; therefore an important direction for research could be to devise algorithms which find a good approximate solution.

The problem of finding centers and medians in weighted networks is currently being investigated [7]. Preliminary results show that most of the algorithms presented here extend in a natural way to this more general case.

REFERENCES

[1] Abram, J.M., and Rhodes, I.B. A decentralized shortest path algorithm, Proc. 16th Allerton Conf. on Communication, Control, and Computing, 1978.

[2] Aburdene, M.F. Numbering the nodes of a graph by distributed algorithms, Proc. 17th Allerton Conf. on Communication, Control, and Computing, 1979.

[3] Angluin, D. Local and global properties in networks of processors, Proc. 12th ACM Symp. on Theory of Computing, 1980.

[4] Bondy, J.A., and Murty, U.S.R. Graph Theory With Applications, MacMillan, London, 1976.

[5] Chang, E.J. Decentralized algorithms in distributed systems. Tech. Rep. CCRG-103, Computer Systems Research Group, University of Toronto, 1979.

[6] Gallager, R.G., Humblet, P.A., and Spira, P.M. A distributed algorithm for minimum weight spanning trees, Res. Rep. LIDS-P-906-A, Mass. Inst. Tech., Laboratory for Information and Decision Systems, 1979.

[7] Korach, E., Rotem, D., and Santoro, N. Distributed algorithms for finding centers and medians in networks: the weighted case, in preparation.

[8] Lynch, N.A. Fast allocation of nearby resources in a distributed system, Proc. 12th ACM Symp. on Theory of Computing, 1980.

[9] Moore, E.F. The shortest path through a maze, Proc. Int. Symp. on Theory of Switching, 1959.

[10] Ramirez, R.J., and Santoro, N. Distributed control of updates in multiple-copy databases: a time optimal algorithm, Proc. 4th Berkeley Conf. on Distributed Data Management and Computer Networks, 1979.

[11] Romani, F. Cellular automata synchronization, Information Sciences, 10, 3 (1976), pp. 299-318.

[12] Santoro, N. Determining topology information in distributed networks, Proc. 11th Southeastern Conf. on Combinatorics, Graph Theory and Computing, 1980.

[13] Slater, P.J., Hedetniemi, S.T., and Cockayne, E.J. Information dissemination in trees, Tech. Rep. CS-TR-78-11, Computer Science Department, University of Oregon, 1978.

[14] Tajibnapis, W.D. The design of a topology information maintenance scheme for a distributed computer network, Proc. ACM Conference, 1974.

[15] Tajibnapis, W.D. A correctness proof of a topology information maintenance protocol for a distributed computer network, CACM, 20, 7 (July 1977) pp. 477-485.

[16] Wall, D.W., and Owicki, S.S. Center-based broadcasting, to appear 1980.

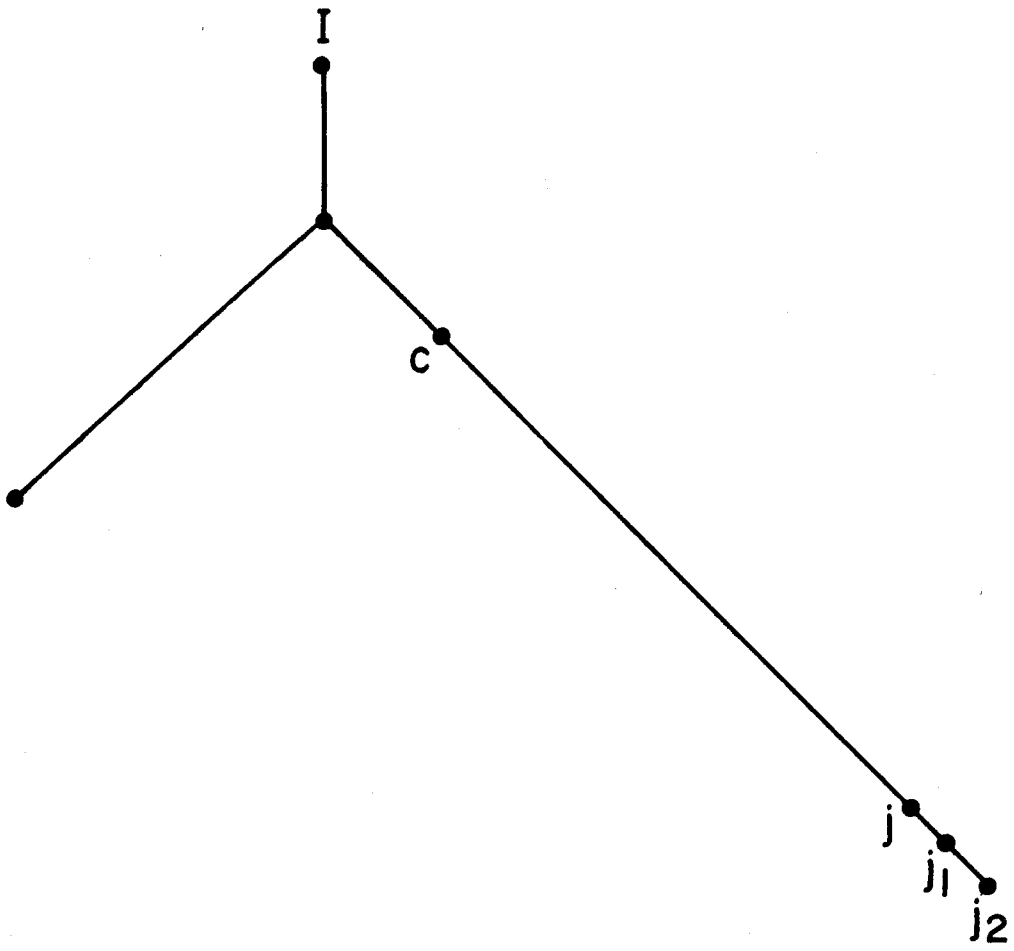

FIG. 1 The addition of the path $<j, j_1, j_2>$ to the tree alters the location of the center; this shows that the information from every node must be taken into account to exactly determine the center.

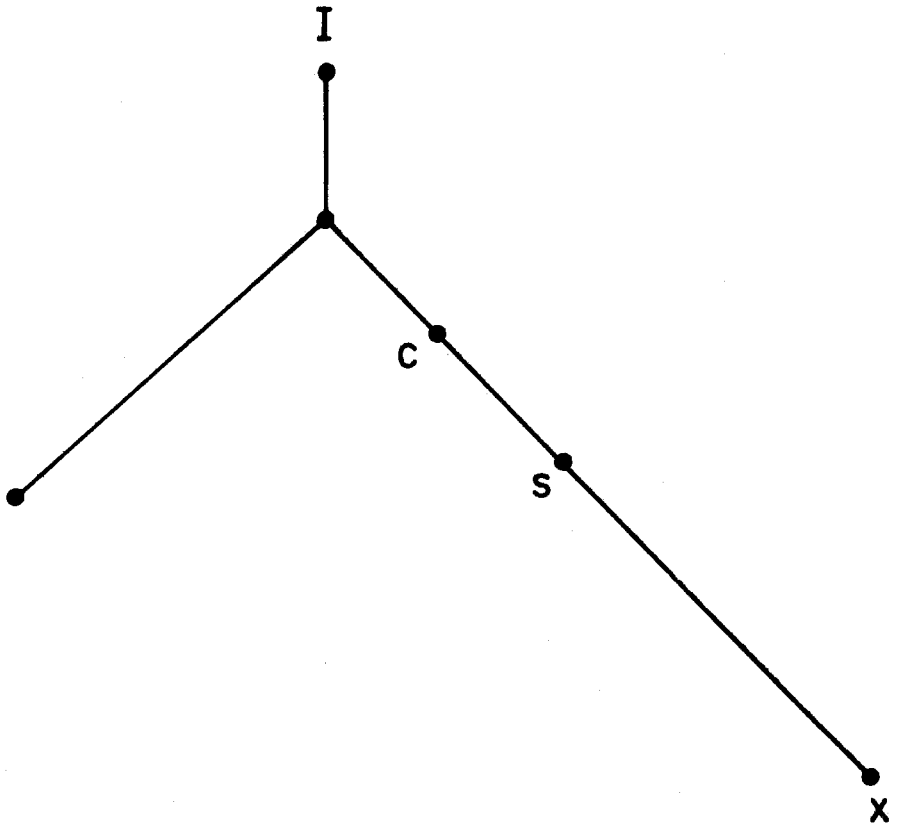

FIG. 2 Each meeting point s in the algorithm TREE-CENTER lies between a center c and a node x farthermost from the initiator I.

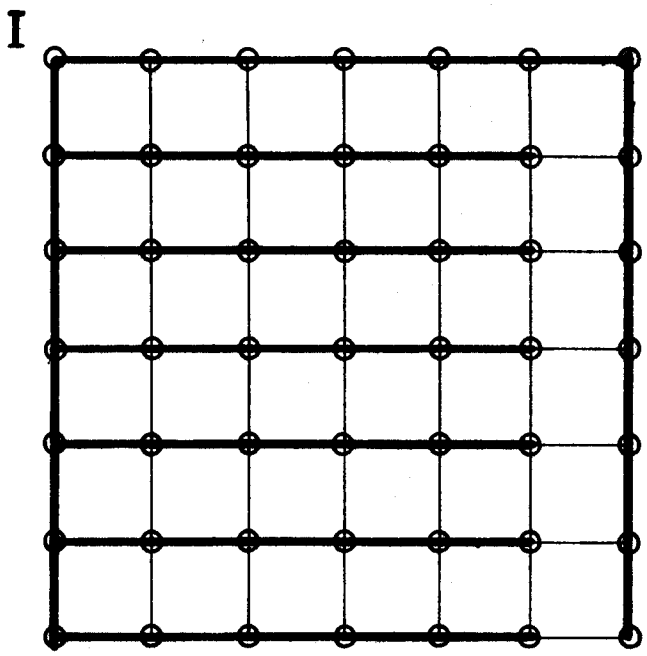

FIG. 3 Node I is a center of the tree $SPT(I)$ (shown in bold) of the $n \times n$ square grid graph G. The radius $r(SPT(I)) = 2(n-1)$ while $r(G) = n-1$.

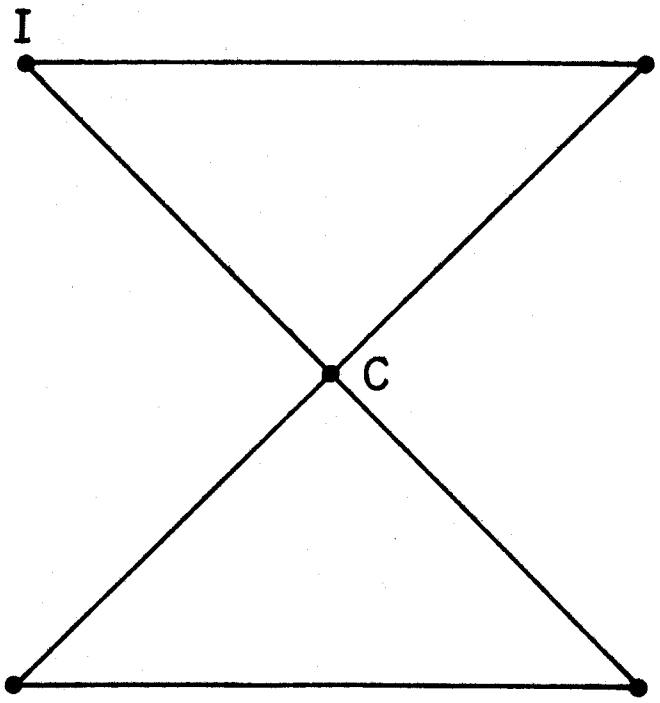

FIG. 4 A simple graph for which algorithm GRAPH-CENTER achieves its upper bound.

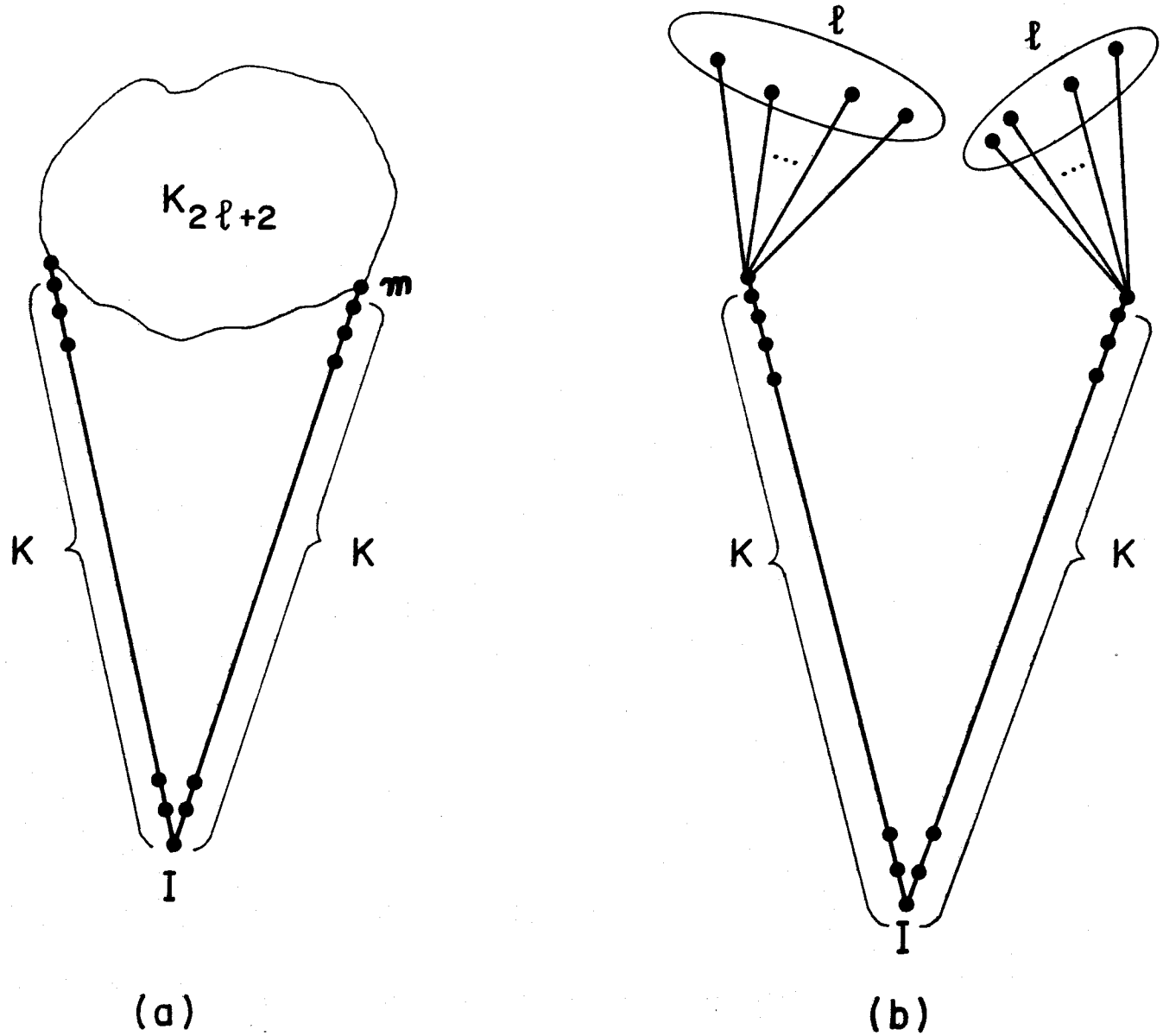

FIG. 5 (a) Family of graphs $G(k)$

(b) A possible tree $SPT(I)$

Fig 5

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WATERLOO
TECHNICAL REPORTS 1979

Report No.	Author	Title
CS-79-01*	E.A. Ashcroft W.W. Wadge	Generality Considered Harmful - A Critique of Descriptive Semantics
CS-79-02*	T.S.E. Maibaum	Abstract Data Types and a Semantics for the ANSI/SPARC Architecture
CS-79-03*	D.R. McIntyre	A Maximum Column Partition for Sparse Positive Definite Linear Systems Ordered by the Minimum Degree Ordering Algorithm
CS-79-04*	K. Culik II A. Salomaa	Test Sets and Checking Words for Homomorphism Equivalence
CS-79-05*	T.S.E. Maibaum	The Semantics of Sharing in Parallel Processing
CS-79-06*	C.J. Colbourn K.S. Booth	Linear Time Automorphism Algorithms for Trees, Interval Graphs, and Planar Graphs
CS-79-07*	K. Culik, II N.D. Diamond	A Homomorphic Characterization of Time and Space Complexity Classes of Languages
CS-79-08*	M.R. Levy T.S.E. Maibaum	Continuous Data Types
CS-79-09	K.O. Geddes	Non-Truncated Power Series Solution of Linear ODE's in ALTRAN
CS-79-10*	D.J. Taylor J.P. Black D.E. Morgan	Robust Implementations of Compound Data Structures
CS-79-11*	G.H. Gonnet	Open Addressing Hashing with Unequal-Probability Keys
CS-79-12	M.O. Afolabi	The Design and Implementation of a Package for Symbolic Series Solution of Ordinary Differential Equations
CS-79-13*	W.M. Chan J.A. George	A Linear Time Implementation of the Reverse Cuthill-McKee Algorithm
CS-79-14	D.E. Morgan	Analysis of Closed Queueing Networks with Periodic Servers
CS-79-15*	M.H. van Emden G.J. de Lucena	Predicate Logic as a Language for Parallel Programming
CS-79-16*	J. Karhumäki I. Simon	A Note on Elementary Homorphisms and the Regularity of Equality Sets
CS-79-17*	K. Culik II J. Karhumäki	On the Equality Sets for Homomorphisms on Free Monoids with two Generators
CS-79-18	F.E. Fich	Languages of R-Trivial and Related Monoids

* Out of print - contact author

CS-79-19*	D.R. Cheriton	Multi-Process Structuring and the Thoth Operating System
CS-79-20*	E.A. Ashcroft W.W. Wadge	A Logical Programming Language
CS-79-21*	E.A. Ashcroft W.W. Wadge	Structured LUCID
CS-79-22	G.B. Bonkowski W.M. Gentleman M.A. Malcolm	Porting the Zed Compiler
CS-79-23*	K.L. Clark M.H. van Emden	Consequence Verification of Flow-charts
CS-79-24*	D. Dobkin J.I. Munro	Optimal Time Minimal Space Selection Algorithms
CS-79-25*	P.R.F. Cunha C.J. Lucena T.S.E. Maibaum	On the Design and Specification of Message Oriented Programs
CS-79-26*	T.S.E. Maibaum	Non-Termination, Implicit Definitions and Abstract Data Types
CS-79-27*	D. Dobkin J.I. Munro	Determining the Mode
CS-79-28	T.A. Cargill	A View of Source Text for Diversely Configurable Software
CS-79-29	R.J. Ramirez F.W. Tompa J.I. Munro	Optimum Reorganization Points for Arbitrary Database Costs
CS-79-30*	A. Pereda R.L. Carvalho C.J. Lucena T.S.E. Maibaum	Data Specification Methods
CS-79-31*	J.I. Munro H. Suwanda	Implicit Data Structures for Fast Search and Update
CS-79-32*	D. Rotem J. Urrutia	Circular Permutation Graphs
CS-79-33*	M.S. Brader	PHOTON/532/Set - A Text Formatter
CS-79-34	D.J. Taylor D.E. Morgan J.P. Black	Redundancy in Data Structures: Improving Software Fault Tolerance
CS-79-35	D.J. Taylor D.E. Morgan J.P. Black	Redundancy in Data Structures: Some Theoretical Results
CS-79-36	J.C. Beatty	On the Relationship between the LL(1) and LR(1) Grammars
CS-79-37	E.A. Ashcroft W.W. Wadge	R_x for Semantics

* Out of print - contact author

CS-79-38	E.A. Ashcroft W.W. Wadge	Some Common Misconceptions about LUCID
CS-79-39*	J. Albert K. Culik II	Test Sets for Homomorphism Equivalence on Context Free Languages
CS-79-40	F.W. Tompa R.J. Ramirez	Selection of Efficient Storage Structures
CS-79-41*	P.T. Cox T. Pietrzykowski	Deduction Plans: A Basis for Intelligent Backtracking
CS-79-42	R.C. Read D. Rotem J. Urrutia	Orientations of Circle Graphs

* Out of print - contact author

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WATERLOO

RESEARCH REPORTS 1980

Report No.	Author	Title
CS-80-01	P.T. Cox T. Pietrzykowski	On Reverse Skolemization
CS-80-02	K. Culik II	Homomorphisms: Decidability, Equality and Test Sets
CS-80-03	J. Brzozowski	Open Problems About Regular Languages
CS-80-04	H. Suwanda	Implicit Data Structures for the Dictionary Problem
CS-80-05	M.H. van Emden	Chess-Endgame Advice: A Case Study in Computer Utilization of Knowledge
CS-80-06	Y. Kobuchi K. Culik II	Simulation Relation of Dynamical Systems
CS-80-07	G.H. Gonnet J.I. Munro H. Suwanda	Exegesis of Self-Organizing Linear Search
CS-80-08	J.P. Black D.J. Taylor D.E. Morgan	An Introduction to Robust Data Structures
CS-80-09*	J.Ll. Morris	The Extrapolation of First Order Methods for Parabolic Partial Differential Equations II
CS-80-10+	N. Santoro H. Suwanda	Entropy of the Self-Organizing Linear Lists
CS-80-11	T.S.E. Maibaum C.S. dos Santos A.L. Furtado	A Uniform Logical Treatment of Queries and Updates
CS-80-12	K.R. Apt M.H. van Emden	Contributions to the Theory of Logic Programming
CS-80-13*	J.A. George M.T. Heath	Solution of Sparse Linear Least Squares Problems Using Givens Rotations
CS-80-14	T.S.E. Maibaum	Data Base Instances, Abstract Data Types and Data Base Specification
CS-80-15	J.P. Black D.J. Taylor D.E. Morgan	A Robust B-Tree Implementation
CS-80-16	K.O. Geddes	Block Structure in the Chebyshev-Padé Table
CS-80-17	P. Calamai A.R. Conn	A Stable Algorithm for Solving the Multi-facility Location Problem Involving Euclidean Distances

* Out of print, contact author

+ In preparation

CS-80-18	R.J. Ramirez	Efficient Algorithms for Selecting Efficient Data Storage Structures
CS-80-19	D. Therien	Classification of Regular Languages by Congruences
CS-80-20	J. Buccino	A Reliable Typesetting System for Waterloo
CS-80-21	N. Santoro	Efficient Abstract Implementations for Relational Data Structures
CS-80-22	R.L. de Carvalho T.S.E. Maibaum T.H.C. Pequeno A.A. Pereda P.A.S. Veloso	A Model Theoretic Approach to the Theory of Abstract Data Types and Data Structures
CS-80-23	G.H. Gonnet	A Handbook on Algorithms and Data Structures
CS-80-24	J.P. Black D.J. Taylor D.E. Morgan	A Case Study in Fault Tolerant Software
CS-80-25	N. Santoro	Four O(n**2) Multiplication Methods for Sparse and Dense Boolean Matrices
CS-80-26	J.A. Brzozowski	Development in the Theory of Regular Languages
CS-80-27	J. Bradford T. Pietrzykowski	The Eta Interface
CS-80-28	P. Cunha T.S.E. Maibaum	Resource = Abstract Data Type Data + Synchronization ...
CS-80-29	K. Culik II Arto Salomaa	On Infinite Words Obtained by Interating Morphisms
CS-80-30	T.F. Coleman A.R. Conn	Nonlinear Programming via an Exact Penalty Function: Asymptotic Analysis
CS-80-31	T.F. Coleman A.R. Conn	Nonlinear Programming via an Exact Penalty Function: Global Analysis
CS-80-32	P.R.F. Cunha C.J. Lucena T.S.E. Maibaum	Message Oriented Programming - A Resource Based Methodology
CS-80-33	Karel Culik II Tero Harju	Dominoes Over A Free Monoid
CS-80-34+	K.S. Booth	Dominating Sets in Chordal Graphs
CS-80-35	Alan George J. W-H Liu	Finding Diagonal Block Envelopes of Triangular Factors of Partitioned Matrices
CS-80-36	D.J. Taylor	Robust Storage Structures for Data Structures

+ In preparation
* Out of print, contact author

Research Reports 1980

CS-80-37	R.B. Simpson	A Two Dimensional Mesh Verification Algorithm
CS-80-38†	D.Rotem J. Urrutia	Finding Maximum Qliques in Circle Graphs
CS-80-39†	S.T. Vuong D.D. Cowan	Automated Validation of a Protocol: The CCITT Recommendation X.75 packet level
CS-80-40	F. Mavaddat	Another Experiment with Teaching of Programming Languages
CS-80-41†	K. Culik II J. Pachl	Equivalence problems for mapping on infinite strings
CS-80-42†	J.A. George E. Ng	A comparison of some methods for solving sparse linear least squares problems
CS-80-43†	T.S.E. Maibaim P.R.F. Cunha	Synchronization calculus for message oriented programming

† In Preparation