CIS-TR-80-12

A LINEAR ALGORITHM FOR FINDING THE CENTER OF A UNICYCLIC GRAPH[#]

by

Sandra Mitchell Hedetniemi*

Stephen Hedetniemi*

Terry Beyer*

Abstract

A unicyclic graph G = (V,E) with N vertices is a connected graph which has exactly one cycle (or equivalently, is connected and has N edges). These graphs occur frequently as the topology of computer or switching networks with ring structures. In this paper an O(N) algorithm is presented for finding the center of any unicyclic graph with N vertices. This algorithm uses a divide-and-conquer strategy, along with a bucket (or radix) sort and a system of pointers to avoid the $O(N^2)$ to $O(N^3)$ computations normally required to find the center of an arbitrary graph.

Research supported in part by the National Science Foundation under Grant MCS-79-03913

* Department of Computer and Information Science, University of Oregon, Eugene, OR 97403

1. Introduction

The notion of centrality in a graph $G = (V,E)$ has numerous real world applications. These typically involve problems of finding optimum locations for such things as medical centers, warehouses, stores, police stations or schools. In general, a "center" is a vertex (or set of vertices) which minimizes some function involving the distance between an arbitrary vertex and a vertex in the center.

For example, one may want to find a vertex x which minimizes a sum such as

$$f(x) = \Sigma d(v,x) \qquad v \in G$$

or a weighted sum such as

$$g(x) = \Sigma a(v)d(v,x) \qquad v \in G, \text{ or}$$

$$h(x) = \Sigma(a(v) + d(v,x)\ ) \qquad v \in G,$$

where $a(v)$ is a non-negative, real-valued weight associated with a vertex v and $d(v,x)$ is the shortest distance in G from v to x.

Alternately, one may want to find a minimax, i.e. a vertex (or set of vertices) which minimizes a maximum such as

$$\max\ \{d(v,x)\},$$

$$\max\ \{a(v)d(v,x)\}, \text{ or}$$

$$\max\ \{a(v) + d(v,x)\}, \text{ for all } v \in G.$$

A represenative sample of papers which use these notions of centrality can be found in [1], [2], [4-11].

In this work we define the (Jordan) <u>center</u> of a graph to consist of the set of all vertices x for which $\max\ \{d(v,x)\}$ is a minimum. The first result which characterized the centers of a given class of graphs was the following due to Jordan in 1869 [8].

<u>Theorem 1</u>. (Jordan) If T is a tree, then the center of T consists of either a single vertex or two adjacent vertices.

The centers of several other classes of graphs have only recently been characterized, including 2-trees [11], $C_{(N)}$-trees [7], unicyclic graphs and cacti [10].

There also exist linear time algorithms for determining the centers of trees [9], maximal outerplanar graphs and 2-trees [4]. In this paper we extend the development of linear algorithms for locating the centers of graphs to include unicyclic graphs, a class of graphs which occurs frequently in the study of computer networks [3].

2. Definitions

A <u>unicyclic graph</u> $G = (V,E)$ is a connected graph which has exactly one cycle C; the vertices of $C = \{v_1, v_2, \ldots, v_k\}$ are called <u>cycle vertices</u>. The <u>branch of cycle vertex v</u> is the subtree of G rooted at v, denoted T_v, (possibly empty), which contains no other cycle vertices than v.

Assume that the cycle C of a unicyclic graph has been given an orientation, i.e. C is a directed cycle of length k. Let $h = \lfloor k/2 \rfloor$. Then the <u>range</u> of cycle vertex v_i, denoted $\mathrm{RANGE}(v_i)$ is the set of h vertices $\{v_{i+1}, v_{i+2}, \ldots, v_{i+h}\}$ following v_i half way around the cycle in the direction of the orientation.

The <u>height</u> of cycle vertex v, $h(v)$, equals $\max \{d(u,v): u \in T_v\}$, where $d(u,v)$ is defined to be the length of a shortest path between vertices u and v.

The <u>eccentricity</u> $e(u)$ of a vertex u in G is the maximum length of a shortest path from u to any other vertex, i.e. $e(u) = \max \{d(u,w): w \in G\}$.

A vertex w in G for which $e(w)$ is minimum is a <u>central vertex</u>. The <u>center</u> of G is the set of all central vertices.

The height, $h(v)$, of a cycle vertex v in a unicyclic graph can be used to refine the definition of the eccentricity of v. For every cycle vertex v, $e(v) = \max \{h(v), d(u,v) + h(u)\}$, where u is a cycle vertex, i.e. there exists a vertex farthest from v either in its branch or in a branch of another cycle vertex u.

For all vertices $w \notin C$, denoted as tree vertices, $e(w) = \max \{d(x,w), d(v,w) + d(u,v) + h(u)\}$, where w and x are in T_v (the branch at v) and u is another cycle vertex. That is, the vertex farthest from vertex w is either in the same branch as w or in a branch of another cycle vertex.

3. A linear representation for unicyclic graphs

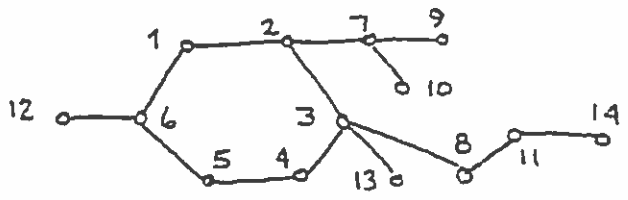
All unicyclic graphs with n vertices can be constructed recursively as follows:

(i) construct a cycle C of length $k \leq n$

(ii) perform the following operation n-k times:

add a new vertex and join it to an existing vertex.

Figure 1 illustrates a linear (implicit) representation of a unicyclic graph which can be generated by this process. Note that any unicyclic graph can be numbered so that the cycle vertices are numbered 1, 2, ..., k. Thus, if the cycle has M vertices, then the first M entries in the representation can denote the cycle. The remainder of the entries describe the branches. It should be noted that such a representation for a unicyclic graph G can be obtained in linear time from a list of the edges of G by using a depth-first search.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
UNICYCLE	6	1	2	3	4	5	2	3	7	7	8	6	3	11

Figure 1

4. Calculation of the height of tree and cycle vertices

In order to determine the center of a unicyclic graph we first need to calculate the eccentricity and height of each vertex. In order to do this we will associate with each tree vertex w a height (HT(w)) which equals the maximum distance from w to any endvertex (vertex of degree 1) in its subtree T_v (as determined by its cycle vertex root v). We will also associate with w the number of its subbranches having that height, NO_HT(w), as well as the height of its second longest branch, NEXT_HT(w).

In [9], a linear algorithm was presented which calculates these heights. This algorithm can be extended to calculate the heights and "second heights" of all the tree vertices (since the calculations are dependent only on the respective branches) by using the last N-M entries in the linear representation of a unicyclic graph with N vertices and cycle length M. We present this extension as Algorithm BRANCHES. It should be noted that after the completion of Algorithm BRANCHES, h(v), the height of cycle vertex v, will have been calculated for all cycle vertices as HT(v).

Algorithm BRANCHES. To find the height HT(V) of each vertex V in a unicyclic graph UNICYCLE [1 .. N] having cycle [1 .. M]; to determine for each tree vertex V the number of branches having HT(V); and to determine the height of the second longest branch NEXT_HT(V) of each vertex.

```
0.  [Initialize]
    for V ← 1 to N do
        HT(V) ← 0
        NO_HT(V) ← 1
        NEXT_HT(V) ← 0
        od
```

```
1.  [Iteratively determine heights]
    for V ← N downto M+1 do
       PARENT ← UNICYCLE(V)
       if HT(PARENT) < HT(V) + 1
            then NEXT_HT(PARENT) ← HT(PARENT)
                 HT(PARENT) ← HT(V) + 1
                 NO_HT(PARENT) ← 1
            else if HT(PARENT) = HT(V) + 1
                      then NO_HT(PARENT) ← 2
                           NEXT_HT(PARENT) ← HT(PARENT)
                      else if HT(V) + 1 > NEXT_HT(PARENT)
                                then NEXT_HT(PARENT) ← HT(V) + 1
                                fi
                      fi
            fi
       od
       stop.
```

5. Calculation of cycle eccentricities

As mentioned earlier, the eccentricity of a cycle vertex v is defined as $e(v) = \max \{h(v), d(v,u) + h(u)\}$, where u ranges over all cycle vertices. After the completion of Algorithm BRANCHES the values $h(v)$ will have been computed for all cycle vertices v. In this section we present a fairly complex, but linear, algorithm which can be used to determine for each cycle vertex v, $\max \{d(u,v) + h(u)\}$ over every other cycle vertex u. In particular, Algorithm CYCLE_ECCENTRICITY determines the value CYCLE_E(v) = $\max \{d(u,v) + h(u)\}$ for every $u \in$ RANGE(v), for every cycle vertex v. A second application of Algorithm CYCLE_ECCENTRICITY with the cycle in the reverse orientation will suffice to determine a second CYCLE_E'(v), in terms of the remaining cycle vertices u not in the original RANGE(v).

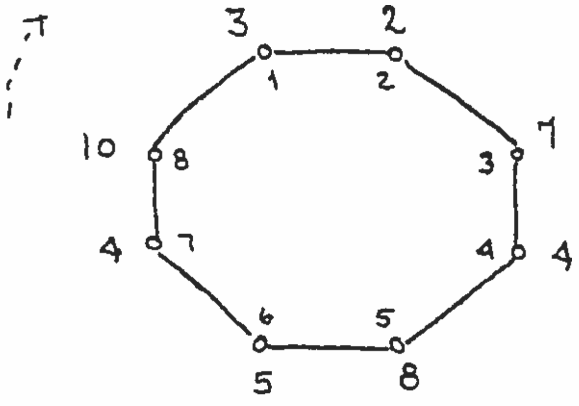
Given these three values, the eccentricity of v is simply e(v) = max {h(v), CYCLE_E(v), CYCLE_E'(v)}.

The primary reason why the center of a unicyclic graph can be found in linear time lies in the fact that all of the values CYCLE_E(v) can be found in linear time. Thus, Algorithm CYCLE_ECCENTRICITY should be presented with some care.

Consider Figure 2, which illustrates the cycle of a unicyclic graph with the cycle vertices numbered 1 to 8 and with each vertex labelled with its height, as determined by Algorithm BRANCHES. Beneath the cycle we have listed the vertices in order, followed by an additional half-cycle since the repeated vertices 1, 2, 3, 4 lie within the indicated clockwise range of vertex 8. Beneath these vertices we have listed their heights; and beneath this we have listed their ADDED_HT, which equals the sum of the height plus the index of each vertex. These ADDED_HT are very useful in computing the cycle eccentricities. Notice for example that ADDED HT(8) - 5 = d(5,8) + h(8), i.e. d(u,v) + h(u) = ADDED_HT(u) - v for u ε RANGE(v).

Notice also that since 18 is the largest value of an ADDED_HT in Figure 2, which occurs for example at vertex 8, we can immediately conclude that the cycle eccentricities of vertices 4, 5, 6 and 7 must be 14, 13, 12 and 11, respectively, since vertex 8 lies within the (clockwise) range of vertices 4, 5, 6 and 7.

It is this important observation that permits us to calculate the cycle eccentricities in O(N) time instead of the anticipated $O(N^2)$ time. For all we need to do is to bucket sort into decreasing order ADDED_HT values (into at most 2N buckets), and to determine the cycle eccentricities once and for all from this list of ADDED_HTs.



								1	2	3	4	
1	2	3	4	5	6	7	8	9	10	11	12	v
3	2	7	4	8	5	4	10	3	2	7	4	h(v)
4	4	10	8	13	11	11	18	12	12	18	16	added-ht(v)
			14	13	12	11	10	9	8			cycle-ecc(v)
12	11	10	9									

Figure 2

Notice furthermore two additional consequences of this ordering of ADDED-HTs. First, vertex 11 (a copy of vertex 3) also has a maximum ADDED-HT of 18, from which we can infer cycle eccentricities of 11, 10, 9 and 8 for vertices 7, 8, 9 and 10, respectively. However, we are only interested in vertices 1 to 8 and can ignore the cycle eccentricity values for vertices 9 and 10.

Second, having processed vertices 11 and 8, each having ADDED-HTs of 18, we could next process vertex 5 having ADDED-HT of 13. From this we would normally conclude that the cycle eccentricities of vertices 1, 2, 3 and 4 were 12, 11, 10 and 9, respectively. However, the cycle eccentricity for vertex 4 had been previously determined (by vertex 8) to be 14. Thus, in processing vertex 5 we need to know which vertices within its range have already had their cycle eccentricities determined.

We can do this using a simple scheme of pointers and the observation that at any time the vertices whose cycle eccentricities have been determined occur in consecutive blocks of length at least $\lfloor M/2 \rfloor$. In our example, after vertex 11 has been processed, the block of completed cycle eccentricities include vertices 7, 8, 9 and 10. After vertex 8 has been processed, this block includes vertices 4 - 10, and finally after vertex 5 has been processed, this block of determined cycle eccentricities includes vertices 1 - 10 (and the algorithm is essentially finished).

In order to calculate the cycle eccentricities, in addition to the linear array ADDED_HT, we will require four linear arrays. The length of all five of these arrays will be $M + \lfloor M/2 \rfloor$; a length sufficient to contain the cycle vertices with a half-cycle "wrap-around". We will assume, for the remainder of the discussion and for the details of the algorithm, a clockwise orientation from v_i to v_{i+1} for all $i > 1$ and from v_M to v_1.

The array ADDED_HT is defined as follows: for $1 \leq I \leq M$, ADDED_HT(I) = $HT(v_I) + I$; and for $M \leq I \leq M + \lfloor M/2 \rfloor$, ADDED_HT(I) = $HT(v_{I-M}) + I$, where HT(I) is obtained from Algorithm BRANCHES.

The second array, FIRST(I), is used to denote the smallest index of any vertex in whose range vertex I occurs. For each I the value of FIRST(I) (recalling the clockwise orientation) is $\max \{1, I - \lfloor M/2 \rfloor \}$.

The third array, CYCLE_E(I), will contain the calculated value of the cycle eccentricity for each vertex I.

The remaining two arrays prevent more than one cycle eccentricity from being calculated for each cycle vertex and insure that all calculations will be done efficiently. The first array, RIGHT_END, contains pointers into the second, LEFT_END, which also contains pointers. The two pointers LEFT_END(I) and RIGHT_END(I) are used to encompass a block of vertices, including I, which are known to already have been assigned a cycle eccentricity. Initially, RIGHT_END(I), for all I, will be set to zero to serve as a flag.

In order that the largest cycle eccentricity is calculated for each cycle vertex during the first (and only) calculation, the values in ADDED_HT will be sorted in descending order and the vertices processed according to this order.

Let vertex I be the next vertex to be processed. Either there exist vertices between FIRST(I) and I-1 that have not had their cycle eccentricities calculated or there do not. The vertices within these boundaries which have not had their cycle eccentricities calculated exist as a single consecutive block.

The determination of whether or not there are any such vertices involves checking the pointers LEFT_END(I-1) and RIGHT_END(I-1). If RIGHT_END(I-1) = 0, then the cycle eccentricity of vertex I-1 has not been calculated. Calculation continues leftward (toward lower I) until a vertex K is reached such that either $K < FIRST(I)$ or $RIGHT\_END(K) \neq 0$. If RIGHT_END(I-1) = J, for example, then LEFT_END(J) indicates a leftmost vertex in a block of vertices containing vertex I-1 for which cycle eccentricities have been calculated. Two possible situations arise, depending on the values LEFT_END(J), FIRST(I) and RIGHT_END(LEFT_END(J) - 1) $\neq$ 0, then no cycle eccentricities will be calculated using vertex I. Otherwise, the cycle eccentricities of the vertices W, $FIRST(I) \leq W \leq LEFT\_END(J) - 1$ will be calculated as CYCLE_E(W) = ADDED_HT(I) - W.

The pointers in RIGHT_END and LEFT_END will be updated to reflect the new knowledge of the calculated cycle eccentricities.

We next present Algorithm CYCLE_ECCENTRICITY which calculates the cycle eccentricity of each cycle vertex v. It is phrased in terms of a clockwise orientation of the cycle which directs each cycle vertex I to its neighbor I+1 and vertex M to 1. Obvious changes are necessary for the counterclockwise orientation.

Algorithm CYCLE_ECCENTRICITY. Given an oriented cycle with heights HT(I) associated with each vertex I, having M vertices; the orientation is assumed to be clockwise directing vertex I to I+1 and vertex M to 1; to calculate the cycle eccentricity CYCLE_E(I) of each vertex I.

```
0.  [Initialize]
    for I ← 1 to M do
       ADDED_HT(I) ← HT(I) + I
       FIRST(I) ← max {1, I - ⌊M/2⌋ }
       RIGHT_END(I) ← 0
       od
    for I ← M+1 to  M/2 + M do
       ADDED_HT(I) ← HT(I-M) + I
       FIRST(I) ← I - ⌊M/2⌋ + 1
       RIGHT_END(I) ← 0
       od
1.  [Sort the vertices according to ADDED_HT]
    Bucket (radix) sort the vertices in decreasing order according to ADDED_HT
    into array LIST
2.  [Process the vertices in LIST]
    for J ← 1 to M + ⌊M/2⌋ do
        I ← LIST(J)
    if RIGHT_END(I-1) = 0
        then START ← I - 1
        else START ← LEFT_END( RIGHT_END(I-1) ) - 1
        fi
    LOOP ← FALSE
    W ← START
    while W ≥ FIRST(I) and RIGHT_END(W) = 0 do
        CYCLE_E(W) ← ADDED_HT(I) - W
        LOOP ← TRUE
        W ← W - 1
        od
    if W < FIRST(I) and LOOP
        then LEFT_END(START) ← W + 1
        else  if LOOP
                  then LEFT_END(START) ← LEFT_END(W)
                  fi
        fi
    od
stop.
```

6. Calculation of the center of a unicyclic graph

After applying Algorithm BRANCHES once and Algorithm CYCLE_ECCENTRICITY twice, (using the two different orientations), to a unicyclic graph G, we are in a position to finish the calculations necessary to determine the center of G. The calculations of the eccentricities of both the cycle vertices and the tree vertices are only partially completed.

The eccentricity of each cycle vertex is the largest of: the two cycle eccentricity values calculated by Algorithm CYCLE_ECCENTRICITY and the value

HT.

The determination of the eccentricities of the tree vertices of a unicyclic graph is a bit involved. We are given, by Algorithm BRANCHES, the height HT(W) of each tree vertex W. From Algorithm CYCLE_ECCENTRICITY we know the eccentricities of the root v of each branch T_v of the unicyclic graph. By working down each branch from this root toward the endvertices we can then determine the eccentricity of each tree vertex.

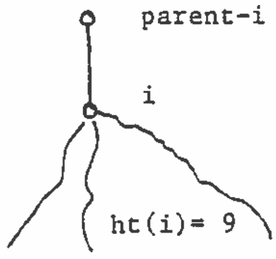
In particular, we can determine the eccentricity of tree vertex I, if we know (i) the height (HT(I)) of I; (ii) if there are at least two subbranches of height HT(I), (NO_HT(I)); (iii) the eccentricity of the parent of vertex I, (ECC(PARENT_I)), where PARENT_I = UNICYCLE(I); (iv) if there are at least two branches or paths leaving the parent which have this eccentricity, (NO_ECC(PARENT_I)); and (v) the second longest path from PARENT_I, (NEXT_ECC(PARENT_I)), disjoint from the first.

Figure 3 illustrates the four possible cases that arise in determining the eccentricity of a tree vertex I. Denote a vertex farthest from I as an <u>eccentric of I</u>. In case (a), I is not on the path from PARENT_I to its eccentric. In case (b), PARENT_I has two eccentrics; therefore, I is not on the path to one of them. In case (c), I is on the path to PARENT_I's eccentric; however, PARENT_I has a next-eccentric which is as far from PARENT_I as HT(I).

We next present Algorithm CENTER to find the center of a unicyclic graph.

<u>Algorithm CENTER</u>. To find the CENTRE of a unicyclic graph UNICYCLE[1..N] having the cycle [1..M].

```
0.  [Perform initial calculations]
    execute Algorithm BRANCHES
    execute Algorithm CYCLE_ECCENTRICITY twice using the clockwise and
```



```
(a)  if ECC(PARENT-I) = 11
         then ECC(I) =  12

(b)  if ECC(PARENT-I) = 10
            and NO-ECC(PARENT-I) ≥ 2
         then ECC(I) = 11

(c)  if ECC(PARENT-I) = 10
            and NO-ECC(PARENT-I) = 1
            and NEXT-ECC(PARENT-I) = 9
         then ECC(I) = 10

(d)  otherwise
                ECC(I) = 9
```

Figure 3

counterclockwise orientations of the cycle; storing results in CYCLE_E(I,1) and CYCLE_E(I,2)

```
1.  [Calculate the eccentricities ECC(I) of the cycle vertices I; iteratively
    determine the center]
    MIN ← N
    CENTRE ← ∅
    for I ← 1 to M do
       ECC(I) ← max {HT(I), CYCLE_E(I,1), CYCLE_E(I,2)}
       NEXT_ECC(I) ← second largest of {HT(I), NEXT_HT(I), CYCLE_E(I,1),CYCLE_E(I,2)}
       if NEXT_ECC(I) = ECC(I)
           then NO_ECC(I) ← 2
           else NO_ECC(I) ← 1
           fi
       if ECC(I) < MIN
           then MIN ← ECC(I)
                CENTRE ← {I}
           else if ECC(I) = MIN
                   then CENTRE ← CENTRE ∪ {I}
                   fi
           fi
       od
2.  [Calculate the eccentricities of the tree vertices; iteratively determine
    the center]
    for I ← M+1 to N do
       PARENT_I ← UNICYCLE(I)
```

```
if ECC(PARENT_I) $>$ 1 + HT(I)
    then [Figure 3, case (a)]
         ECC(I) $\leftarrow$ 1 + ECC(PARENT_I)
         NO_ECC(I) $\leftarrow$ 1
         NEXT_ECC(I) $\leftarrow$ HT(I)
    else [ecc(parent-i) = 1 + ht(i)]
         if NO_ECC(PARENT_I) $\geq$ 2
            then [Figure 3, case (b)]
                 ECC(I) $\leftarrow$ 1 + ECC(PARENT_I)
                 NO_ECC(I) $\leftarrow$ 1
                 NEXT_ECC(I) $\leftarrow$ HT(I)
            else [no-ecc(parent-i) = 1 and ecc(parent-i) = 1 + ht(i)]
                 if NEXT_ECC(PARENT_I) = HT(I)
                    then [Figure 3, case (c)]
                         ECC(I) $\leftarrow$ 1 + HT(I)
                         NO_ECC(I) $\leftarrow$ 1
                         NEXT_ECC(I) $\leftarrow$ HT(I)
                    else [Figure 3, case (d), next-ecc(parent-i) = ht(i)]
                         ECC(I) $\leftarrow$ HT(I)
                         if NEXT_ECC(PARENT_I) = ECC(I) - 1
                                      or NO_HT(I) $\geq$ 2
                            then NO_ECC(I) $\leftarrow$ 2
                            else NO_ECC(I) $\leftarrow$ 1
                            fi
                         NEXT_ECC(I) $\leftarrow$ max {NEXT_HT(I), NEXT_ECC(I)}
                 fi
         fi
    fi
```

```
        if ECC(I) < MIN
           then MIN ← ECC(I)
                CENTRE ← {I}
           else if ECC(I) = MIN
                   then CENTRE ← CENTRE ∪ {I}
                   fi
           fi
        od
    stop.
```
■

7. Complexity of Algorithm CENTER

The intialization step in Algorithm CENTER involves the application of two other algorithms. Algorithm BRANCHES essentially contains two simple loops and is clearly O(N). The analysis of Algorithm CYCLE-ECCENTRICITY should be done more carefully. The initialization is clearly linear in the length of the cycle. The bucket sort can be performed in O(N) time since the possible values for the heights of the cycle vertices certainly must be in the range 0 to N. Each cycle vertex will be processed exactly once from LIST. Because of the value stored in RIGHT-END no vertex will have its cycle eccentricity calculated more than once. Because of the two levels of pointers, the determination of which vertices have already had their cycle eccentricities calculated can be made in constant time. Hence, Algorithm CYCLE-ECCENTRICITY can be performed in O(N) time. Steps 1 and 2 in Algorithm CENTER are clearly linear in the number of vertices. Hence, the center of a unicyclic graph can be determined in time linear in the number of vertices.

8. Bibliography

[1] Adam, A., The centrality of vertices in trees, Studia Sci. Math. Hungar., 9 (1974), no. 3-4, 285 - 303.

[2] Christofides, N., Graph Theory: An Algorithmic Approach, (Academic Press: New York), 1975.

[3] Davies, D., et al, Computer Networks and their Protocols, (Wiley: New York), 1979.

[4] Farley, A. and Proskurowski, A., Computation of the center and diameter of outerplanar graphs, Univ. of Oregon Tech. Rept. CS-TR-79-4, submitted.

[5] Goldman, A., Optimal center locations in simple networks, Trans. Sci., 5 (1971), 212 - 221.

[6] Hakimi, S., Optimum location of switching centres and the absolute centres and medians of a graph I, Ops. Res., 12 (1964), 450 - 459.

[7] Hedetniemi, S. M., Hedetniemi, S. T., and Slater, P., Centers and medians of $C_{(N)}$-trees, in preparation.

[8] Jordan, C., Sur les assemblages de lignes, J. Reine Angew Math., 70 (1869), 185 - 190.

[9] Mitchell, S., Cockayne, E., and Hedetniemi, S., Linear algorithms for finding the Jordan center and path center of a tree, Trans. Sci., to appear.

[10] Mitchell, S., and Hedetniemi, S., Centers of recursive graphs, Univ. of Oregon Tech. Rept. CS-TR-79-11, submitted.

[11] Proskurowski, A., Centers of two-trees, Univ. of Oregon Tech. Rept. CS-TR-79-9, submitted.