CS-TR-77-4

MINIMUM DOMINATING CYCLES IN MAXIMAL OUTERPLANAR GRAPHS

by

Andrzej Proskurowski

Department of Computer Science
University of Oregon, Eugene, OR

Abstract

We consider the class of maximal outerplanar graphs and present a linear time algorithm for finding minimum dominating cycles of such graphs. We stress the use of a particular representation of these graphs called a recursive representation, and some linear operations on binary trees derived from these graphs.

0. Introduction

The literature on the subject of domination is steadily growing. A recent survey of it may be found in [3]. Algorithmically, the problem of finding a minimum dominating set in an arbitrary graph is known to be NP-complete. However, a linear algorithm exists for finding a minimum dominating set in a tree, [2]. This result is extended in [5] to find R-bases of trees. Our paper appears to be the first to define and study dominating cycles in graphs. In addition to this work, an algorithmic study of maximal outerplanar graphs was initiated in [4].

An outerplanar graph is a graph which can be embedded in a plane in such a way that every vertex lies on the exterior face. A maximal outerplanar graph (hereafter called a mop) is an outerplanar graph such that the addition of an edge between any two nonadjacent vertices results in a graph which is not outerplanar. There are several characterizations of mops, two of which are of interest here. The first is that a graph is a mop iff it is isomorphic to a triangulation of a polygon. The second is that a graph is a mop iff it can be constructed from a (base) triangle by a finite number of applications of the following operation: to the graph already constructed add a new vertex in the exterior face and join it to two adjacent vertices on the exterior face. Figure 1a illustrates a mop constructed by this process starting from a base triangle labeled 1,2,3. The i-th vertex for $i>3$ is joined to two vertices with labels less than i.

With any mop we can associate a dual tree, which is obtained by placing a node inside each triangle of the mop and joining two nodes if the corresponding triangles have an edge in common (cf. Figure 1). In this paper we investigate the idea that certain properties of a mop can be determined by examining its dual tree. In particular we are interested in algorithms for solving certain routing problems in mops. For example, what is the length of a shortest closed walk (route) which comes within distance one of every vertex in a mop? Stated more formally, a cycle C in a graph G is a dominating cycle if every vertex not in C is adjacent to at least one vertex in C. A minimum dominating cycle in G has minimum length among all dominating cycles in G. In what follows we will use the dual tree of a mop G to construct a linear algorithm for finding a minimum dominating cycle in G.

1. Representing a mop

In [4] it was suggested that recursive representation of a tree lends itself to the development of very efficient algorithms on trees and that the same efficiencies may be obtained for algorithms on mops using the recursive representation techniques.

Def.1.1. The recursive representation of a mop

Any mop with n vertices and a distinguished base triangle can be represented by two arrays, LOW[1..n] and HIGH[1..n], such that

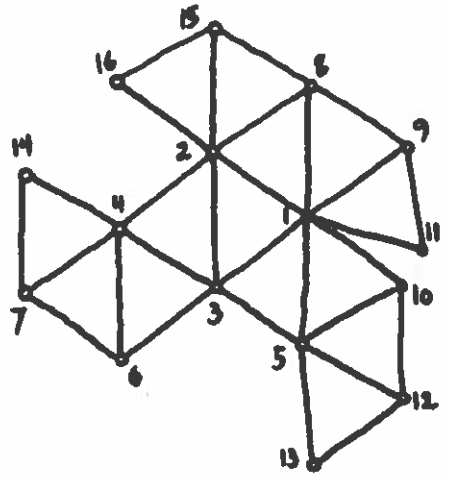
(i)the vertices of the base triangle are labeled 1,2,3;

(ii)for all i ($3 \le i \le n$), i ,LOW[i], and HIGH[i] are the vertices

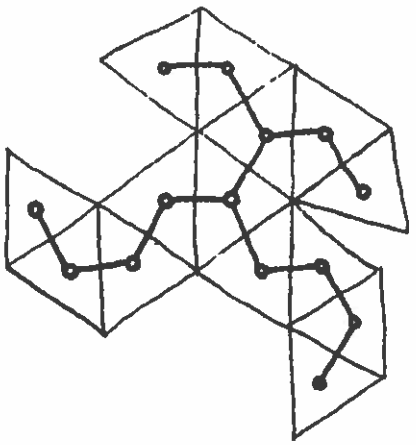
of a triangle where LOW[i]<HIGH[i]<i;

(iii) for vertices 1 and 2 the relation i>HIGH[i] is violated in the obvious way.

(a)



(b)



(c)

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HIGH[i]	3	3	2	3	3	4	6	2	8	5	9	10	12	7	8	15
LOW[i]	2	1	1	2	1	3	4	1	1	1	1	5	5	4	2	2

Figure 1. A mop (a), its dual tree (b), and the recursive representation (c).

Def.1.2. The dual tree of a mop

The dual tree of a mop G is a tree T(G) whose nodes correspond one-to-one with the triangles (meshes) of G, and two nodes are adjacent in T(G) iff the corresponding triangles in G have an edge in common.

Def.1.3. The rooted dual tree of a mop

For any mop G with a base triangle there is a corresponding rooted dual tree which is obtained by rooting T(G) at the node corresponding to the base triangle of G. If we assume that each

node of T(G) is labeled by the highest numbered vertex in the corresponding triangle of G, with the root labeled 3, then the recursive representation of T(G) is given by the array HIGH[3..n] (where entries less than 3 in HIGH are relabeled 3).

In order to distinguish between elements of a mop G and elements of its dual tree T(G), we will use vertices for G and nodes for T(G).

Def.1.4. Base of a triangle

In a mop G with a base triangle, which is represented by arrays LOW and HIGH we will call the edge (LOW[i],HIGH[i]) the base of triangle i ($3<i\leq n$). We call vertices LOW[i] and HIGH[i] the basic vertices of this triangle, and vertex i - the nonbasic vertex of triangle i.

For any mop G, the dual tree T(G) is unique up to an isomorphism. However, there may be more than one mop for which this tree is the dual tree. To remove this ambiguity let us define the dual binary forest.

Def.1.5. Inclination of a node

With each node of a binary tree, except for the root, we can associate an inclination, which indicates whether the node is a left or a right descendant.

Def.1.6. Dual binary forest

The dual binary forest of a mop G with a base triangle is an ordered forest of three rooted binary trees, constructed as follows:

(i) each node of the forest corresponds to a triangle of G (the base triangle is not represented explicitly);

(ii) the roots of these three trees correspond to the triangles adjacent to the base triangle. In case one or more of these triangles does not exist, the corresponding tree is empty;

(iii) in a triangle with vertices labeled 1,2,3, let us call the edge (1,2) a 'right' edge, (2,3) a 'right' edge, and (1,3) a 'left' edge. With this designation, we can proceed to uniquely assign 'left' and 'right' to every other edge of a mop G, with a base triangle labeled 1,2,3. If vertex i is adjacent to vertices LOW[i] and HIGH[i], and (LOW[i],HIGH[i]) is a 'left' edge, then assign 'left' to edge (LOW[i],i) and 'right' to edge (HIGH[i],i). Otherwise, assign 'right' to edge (LOW[i],i) and 'left' to (HIGH[i],i);

(iv) given this assignment of 'left' and 'right' to the edges of a mop G, with a base triangle 1,2,3, we can determine inclination of the nodes in the rooted binary trees of the dual forest of G. If a node j is adjacent to a node i in one of these trees (with i<j), then j is a 'left' ('right') descendant of i if the edge in common between the two corresponding triangles is a 'left' ('right') edge.

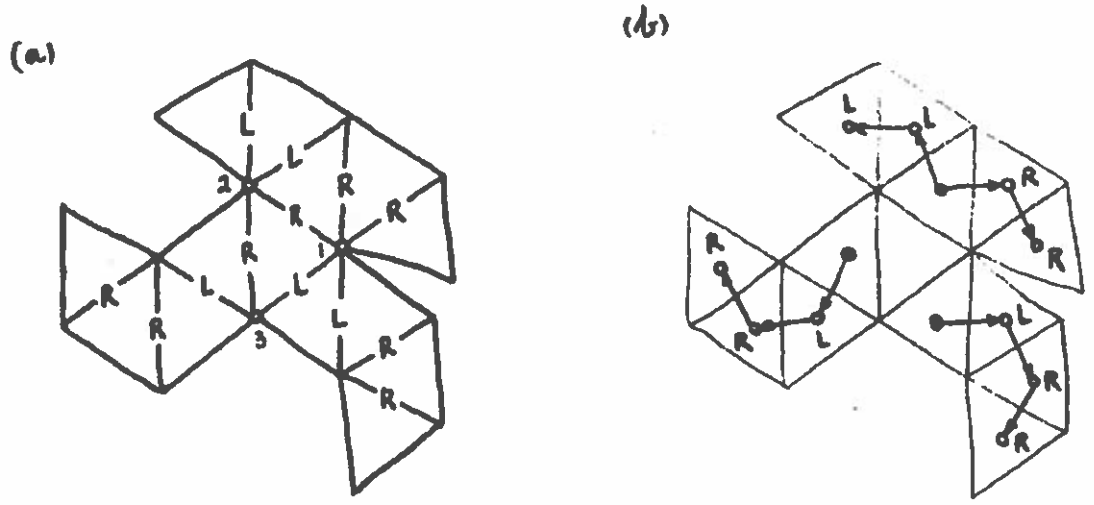


Figure 2. Inclinations of edges in a mop (a) and its dual binary forest (b)

The following algorithm computes the inclination of each node of the dual binary tree of a mop, which is given by a recursive representation in arrays LOW and HIGH.

Algorithm 1.7. Finding inclinations

```
Procedure INCLINATIONS
/* input:  recursive representation of the mop */
        (low,high:array [1..n] of integer;
/* output:the representation of the dual binary forest*/
        var incl :array[3..n] of inclination);
/*local variables describing inclination of base vertices*/
var     lolog,hilog:  array[2..n] of inclination;
        l,h:  inclination;
        i,k,n:integer;
begin /* orient the edges of the base triangle */
                lolog[2]:=right;hilog[2]:=left;
                lolog[3]:=left;hilog[3]:=right;
        /* proceed down the binary trees propagating inclinations */
        for i:=4 to n do
        begin k:=high[i];
              if low[k]=low[i] then begin l:=lolog[k];h:=hilog[k] end
                               else begin l:=hilog[k];h:=lolog[k] end;
                incl[i]:=l;  lolog[i]:=l;  hilog[i]:=h
                end
end;/* inclinations */
```

2. The Hamiltonian cycle of a mop

It is obvious from the definition that every mop has a Hamiltonian cycle. In [1] an algorithm is given for listing this cycle in a mop. Here we give a different algorithm which computes the Hamiltonian cycle of a mop G, given by a recursive representation HIGH and LOW. The cycle is obtained from the dual binary forest of G by noting that for a nonbase triangle t

(i) if t is represented in the forest by a node t which is a leaf then the edges (t,HIGH[t]) and (t,LOW[t]) are in the cycle;

(ii) if t is represented by a node t with only one descendant, s, then the only edge of t in the cycle is non-basic in t and does not belong to triangle s;

(iii) if t is represented by a node with two descendants then none of t's edges are in the Hamiltonian cycle.

The following algorithm links vertices of a mop given by its recursive representation LOW, HIGH and the inclinations array, INCL, into a doubly linked list of the Hamiltonian cycle.

Algorithm 2.1. The Hamiltonian cycle

```
procedure CYCLE
/* input :  representation of the mop and the dual forest */
        (low, high :  array [1..n] of integer;
        incl :  array [3..n] of inclination;
/* output :  double linked list of vertices of the cycle */
        var links :  array [1..2,1..n] of integer);
var     marks :  array [2..n] of integer;
        i,hi,lo:  integer;  inc:  inclination;
        procedure link(u,v,ind:  integer);
        begin links[ind,u]:=v;
                if links[1,v]=0 then links[1,v]:=u
```

```
                                else links[2,v]:=u
                end;/* of linking vertices u and v */
begin /* 0.  initialize marks and links */
     for i:=2 to n do
        begin marks[i]:=0;links[1,i]:=0;  links[2,i]:=0 end;
        links[1,1]:=0;  links[2,1]:=0;
        /* dummy inclination of vertex 3 */ incl[3]:=left;
      /* 1.  scan down:  link and mark */
     for i:=n downto 3 do
        begin hi:=high[i];  lo:=low[i];  inc:=incl[i];
              case marks[i] of
                0:  begin link(i,hi,1);  link(i,lo,2) end;
                1:  if inc=left then link(i,hi,2)
                                else link(i,lo,2);
                2:  if inc=right then link(i,hi,2)
                                else link(i,lo,2);
                3:  end;  /* of cases */
              marks[hi]:=marks[hi]+ord(inc)
              end;  /* of down scan */
       /* 2.  complete linking for the base triangle */
     if marks[2]=0 then link(2,1,2)
     end;  /* of the Hamiltonian cycle */
```

3. Dominating cycle of a mop

We will now present an algorithm for computing a dominating cycle (or, equivalently, a dominating mop) of a given mop. We recall that a dominating set of a graph G is a set of vertices, S, such that every vertex of G not in S is adjacent to a vertex in S.

Def.3.1. Dominating mop

For a given mop G, a dominating mop is a subgraph induced by a cycle of G the vertices of which dominate G.

In order to find a dominating mop of a mop G we will perform a "pruning" of the dual binary forest of G.

Def.3.2. Pruning of a tree

Deleting (pruning) an end-node of a dual binary forest of a mop G corresponds to deleting the corresponding vertex of G.

Fact 3.3. Pruning a dual binary forest of a mop G produces a dual binary forest of a corresponding subgraph of G.

Certain special subgraphs of mop's, entirely dominated by one of their vertices, allow extensive pruning of the corresponding trees.

Def.3.4. Fan

In a mop G, a maximal subgraph F of G such that all triangles of F have one vertex in common and an edge incident to this vertex is the only edge F has in common with other triangles of G is called a fan. The triangle containing this common edge is called the basic triangle of the fan F.

In Figure 3a bold lines indicate fans of the mop from Figure 2.

Def.3.5. Marks

With each node of the dual binary tree of a mop we will associate a mark 'NIL', 'L', 'R', 'L+R', or 'FIXED' as function of inclination and marks of its descendants:

(i)an end-node will be marked 'NIL';

(ii)a node having only a left descendant marked 'NIL' or 'L' will be marked 'L';

(iii)a node having only a right descendant marked 'NIL' or 'R'

will be marked 'R';

(iv)a node with two descendants - left marked 'NIL' or 'L' and right marked 'NIL' or 'R' - will be marked 'L+R';

(v)in all other cases the node will be marked 'FIXED'.

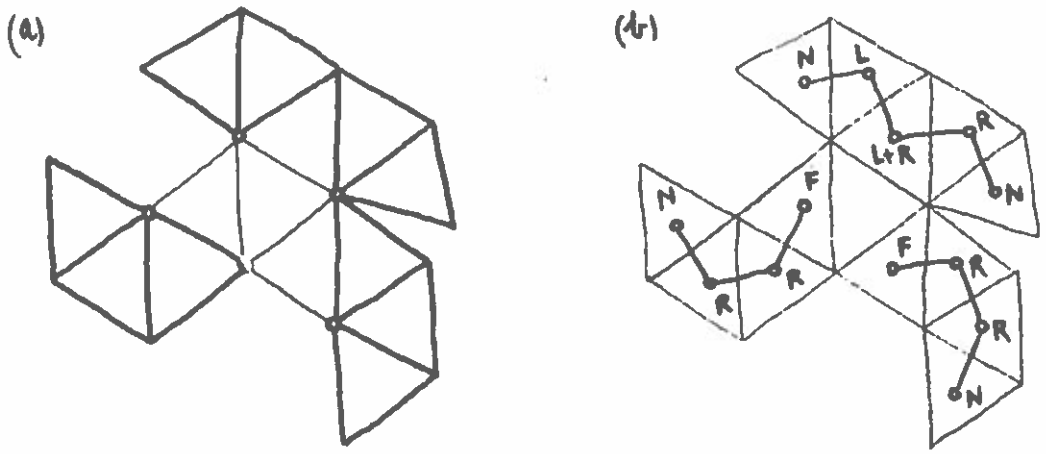


Figure 3. A mop with fans (a) and the marks of the nodes of its dual binary forest (b).

The following classification of the nodes of a dual binary forest of a mop establishes a relation between pruning nodes of the forest (deleting vertices of the mop) and finding a dominating mop.

Fact 3.6. Pruning the dual binary tree

Depending on their out-degrees we can classify the nodes of a dual binary tree as follows:

0. end-node. Clearly, if we delete from a mop G a vertex corresponding to an end-node of the binary tree of G, then resulting mop will dominate all of G;

1.nodes with one descendant. We distinguish three categories of nodes with only one descendant, depending on their marks and inclination:

1.a)mark_and_inclination_agree. The node corresponds to a nonbasic triangle of a fan. We can also delete a vertex corresponding to a nonbasic triangle of a fan and the resulting mop will still be dominating G;

1.b)mark_and_inclination_disagree. The node corresponds to a basic triangle of a fan. A basic vertex of this triangle must be retained in order to dominate all the vertices of the fan;

1.c)marked_'F'. The node is necessary to preserve the connectivity of the dual tree of the dominating mop (the non-separability of the dominating mop);

2.nodes with two descendants. Both basic vertices of the triangle corresponding to a node of out-degree 2 have to be retained to dominate the vertices of descendant triangles. thus the father triangle must not be removed. There are two types of nodes of degree 2:

2.a)marked_'F'. The node must be retained to preserve connectivity of the pruned binary tree;

2.b)marked_'L+R'. Descendants of the node form fans (of one or more triangles) such that the corresponding vertices of the mop are dominated by the basis of the current triangle. Thus the non-basic vertex of this triangle (and the node) can be deleted, and the resulting mop will still be dominating.

In view of Fact 3.6., the following algorithm calculates, for a given mop G, a minimal dominating mop containing the base triangle of G.

Algorithm 3.7. Pruning the dual forest

```
procedure PRUNE/* deletes vertices of the mop */
/* input:  recursive representation of the mop */
       (low,high:  array [1..n] of integer;
/* representation of the dual binary forest */
       incl:  array [4..n] of inclination;
/* output:  the base triangle is implicitely retained*/
       var retain:  array [4..n] of boolean);
/* local variables:  out-degree of the nodes and marks: 0 -- nil, */
var    outval,marks:  array [3..n] of integer;  /*     1 -- left, */
       i,k,n :  integer;                        /*     2 -- right,
                                                       3 -- l + r,
                                              more than 3 -- fixed */
begin  /* initialize out-degrees and marks */
       for i:=4 downto n do begin marks[i]:=0;  outval[i]:=0 end;
/* main loop:  scanning the recursive representation right-to-left,
   deleting vertices and marking fathers accordingly */
       for i:=n downto 4 do
       begin   if high[i]<=3 then k:=3 else k:=high[i];
               case outval[i] of
                0:/* end-node, delete and mark the father */
                begin retain[i]:=false;
                      marks[k]:=marks[k] + ord(incl[i])
                      end;
                1:/* the only son */
                if ord(incl[i])=marks[i]
                then /* fan */begin retain[i]:=false;
                                  marks[k]:=marks[k] + marks[i]
                                  end
                else begin retain[i]:=(marks[i]>3);
                      marks[k]:=4 end;
                2:/* fix unless base of outward fans */
                begin retain[i]:=(marks[i]>3);  marks[k]:=4 end
                end /* of cases */;
               outval[k]:=outval[k] + 1
       end /* of pruning */;
```

4. A minimum dominating cycle of a mop

It is easy to construct examples in which minimum dominating mop of a mop G does not contain the base triangle of G. Consequently, algorithm PRUNE above does not always find a minimum dominating mop. However we can modify it so that it does find a minimum dominating mop, as follows:

1. Find one triangle, t, in a minimal dominating mop of G;
2. Re-label G with t as the base triangle;
3. Apply algorithm PRUNE to get a minimal dominating mop.

The following fact helps to solve the task of 1. above:

Fact 4.1. A triangle of a minimal dominating mop.

For a given mop G, the first triangle retained by algorithm PRUNE belongs to a minimal dominating mop of G.

Proof 4.1. Algorithm PRUNE retains or deletes a triangle, t, only after all descendants of t (relative to the chosen base triangle) have been considered. If t has been the first triangle retained by PRUNE it is necessary to dominate vertices of its descendant triangles, unless there is a dominating mop consisting of one or more of these triangles and not containing t. But then a minimal dominating mop consists of exactly one triangle, and t may serve as such.

The above observations lead to a statement of uniqueness of a minimal dominating mop:

Fact 4.2. If a mop G cannot be dominating by one triangle, then G has a unique minimum dominating mop.

In order to change the base triangle of a recursively represented mop G we must relate the vertices of G in such a way that the vertices of the new base triangle have labels 1, 2, and 3. The following algorithm performs such a "rerooting" of a mop (analogous to "rerooting" a tree).

Algorithm 4.3. Rerooting a mop

```
procedure REROOT/* "rerooting" the mop to new base */
/* input:  recursive representation of the mop */
     (low,high:  array [1..n] of integer;
     newbase :  integer;  /* the new base triangle */
/* output:  recursive representation of the rerooted mop */
     var outlo,outhi:  array [1..n] of integer;
     var xref :  array [1..n] of integer);  /* cross-reference */
var  i,k,n,father,a,b,c,t,second,third:  integer;
begin /* 0.initialize xref and out-rep */
     for i:=1 to n do xref[i]:=0;
     outlo[1]:=2;  outhi[1]:=3;       xref[newbase]:=1;
     outlo[2]:=1;  outhi[2]:=3;       xref[high[newbase]]:=2;
     outlo[3]:=1;  outhi[3]:=2;       xref[low[newbase]]:=3;
/* 1.main loop:  climb up to the root renaming the nodes
              and creating the out-rep */
     father:=newbase;  k:=4;
     while father>3 do
     begin   father:=high[father];    a:=xref[father];
             second:=high[father];    b:=xref[second];
             third:=low[father];      c:=xref[third];
             if c>b then begin b:=c;  c:=second end
                    else c:=third;
             if b>a then begin t:=a;  a:=b;  b:=t end;
             xref[c]:=k;  outlo[k]:=b;  outhi[k]:=a;
             k:=k+1
             end;
/* 2.clean-up:  copy down not affected triangles */
     for i:=father to n do while xref[i]=0 do
     begin   xref[i]:=k;
             second:=xref[high[i]];
             third:=xref[low[i]];
             if second>third
                then begin outhi[k]:=second;outlo[k]:=third end
                else begin outhi[k]:=third;outlo[k]:=second end;
             k:=k+1
             end
     end;/* of rerooting */
```

The complete algorithm for finding the minimal dominating mop of a mop consists of several calls to the above procedures; procedure PRUNE must accomodate request for interruption of computation when the first triangle is 'fixed', and return information about this triangle.

The form of the main program is outlined below.

Algorithm 4.4. Minimal dominating mop

```
   procedure MIN_DOM_MOP
/* input */     (LOW, HIGH :  array [1..n] of integer;
/* output */    var RETAIN :  array [3..n] of boolean;
                var XREF :  array [1..n] of integer);
   var  INCL :  array [4..n] of inclination;
        OUTLO,OUTHI :  array [1..n] of integer;
        newbase :  integer;
   begin
/* 1.Find the dual binary forest */
      INCLINATIONS(LOW, HIGH, INCL);
/* 2.Find a triangle of a minimal dominating mop */
      PRUNE(HIGH, INCL, RETAIN, newbase);
/* 3.Represent the mop with the new base triangle */
      REROOT(LOW, HIGH, newbase, XREF, OUTLO,OUTHI);
      INCLINATIONS(OUTLO,OUTHI, INCL);
/* 4.Find a minimal dominating mop in the rerooted mop */
      PRUNE(OUTHI, INCL, RETAIN);
```

A PASCAL program implementing the above procedure may be found in Appendix A.

5. Efficiency of the algorithm

Algorithm 4.4. requires five scans over the recursive representation of a mop. In each scan a triangle of a mop is processed at most once in a constant amount of time. Thus the algorithm is linear although the number of passes may suggest some

inefficiency. One can easily see that much of the computation in steps 3 and 4 duplicates that of steps 1 and 2. To avoid this waste we have designed an algorithm which, besides steps 1 and 4 (without rerooting) of Algorithm 4.4., requires only a "correction pass" in which the redundant triangles of a dominating mop are discarded.

The information needed by the correction pass is supplied in a (modified) step 1, where the marks indicate "reasons" for tentative retension of a triangle. When the first triangle is "fixed", the pruning is continued after the triangles connecting this "new base" with the old base are doubly linked. These triangles are tentatively retained because of the original choice of the base triangle and therefore the correction pass is limited only to them.

The complete computer program (written in PASCAL) implementing the more efficient algorithm outlined above is listed as Appendix B.

6. Acknowledgment

The author gratefully acknowledges the help of Dr. S. Hedetniemi's constructive criticism of the manuscript.

7. References

[1] Beyer,T, Jones,W, Mitchell,S; A linear algorithm for isomorphism of maximal outer planar graphs (in preparation).

[2] Cockayne,E.J, Goodman,S,E, Hedetniemi,S.T.; A linear algorithm

for the domination number of a tree; Information Processing Letters 4(1975), pp. 41-44.

[3] Cockayne,E.J., Hedetniemi,S.T.; Towards a theory of domination in graphs; Networks, 7(1977),pp. 247-261.

[4] Mitchell,S; Algorithms on trees and maximal outer planar graphs: design, complexity analysis, and data structures studies; PhD Thesis, University of Virginia, 1976.

[5] Slater,P; R-domination in graphs, JACM 23(1976), pp. 446-450.

8. Appendices

A. Procedure MIN_DOM_MOP computing a minimal dominating mop of a mop given by its recursive representation using algorithm 4.4.

B. Procedure MIN_DOM_MOP_EFF computing a minimal dominating mop in two scans of the recursive representation of the given mop.

```
PROCEDURE MIN_DOM_MOP(N: INTEGER;
/*   INPUT: RECURSIVE REPRESENTATION OF THE MOP */
        LOW,HIGH:ARRAY [1..N] OF INTEGER;
/*  OUTPUT: VERTICES IN THE DOMINATING MOP /*
        VAR RETAIN: ARRAY [4..N] OF INTEGER;
        VAR XREF : ARRAY [1..N] OF INTEGER); /* CROSS-REFERENCE */

TYPE  INCLINATION=(NIL,LEFT,RIGHT);
        INTARRAY=ARRAY [1..N] OF INTEGER;
VAR     I,K,N,FATHER,NEWBASE:INTEGER;
        INCL: ARRAY[4..N] OF INCLINATION;
        OUTLO,OUTHI: INTARRAY; /*OUTPUT REP*/

PROCEDURE INCLINATIONS(LOW,HIGH : INTARRAY);
/* SETTING INCLINATION OF NODES INTO 'INCL' */
VAR     /*LOCAL VARIABLES DESCRIBING INCLINATION OF BASE VERTICES*/
        LOLOG,HILOG: ARRAY[2..N] OF INCLINATION;
        L,H: INCLINATION;
   BEGIN LOW[2]:=1;LOLOG[2]:=RIGHT;HILOG[2]:=LEFT;
                LOLOG[3]:=LEFT;HILOG[3]:=RIGHT;
        FOR I:=4 TO N DO
        BEGIN K:=HIGH[I];
                IF LOW[K]=LOW[I] THEN BEGIN L:=LOLOG[K];H:=HILOG[K] END
                                 ELSE BEGIN L:=HILOG[K];H:=LOLOG[K] END;
                INCL[I]:=L; LOLOG[I]:=L; HILOG[I]:=H
                END;
   END;/* INCLINATIONS */

PROCEDURE PRUNE(HIGH:INTARRAY;FLAG:BOOLEAN);
/* PRUNING THE DUAL BINARY FOREST */
VAR     MARKS : INTARRAY; /*MARKS: 0 -- NIL,  1 -- LEFT,
                          2 -- RIGHT,  3 -- L+R,  4 -- FIXED */
BEGIN /*0.INITIALIZE: ZERO  MARKS*/
        FOR I:=4 TO N DO MARKS[I]:=0;
    /*MAIN LOOP: SCANNING THE RECURSIVE REPRESENTATION RIGHT-
        TO-LEFT DELETE END-NODES AND MARK FATHERS ACCORDINGLY*/
        FOR I:=N DOWNTO 4 DO
        BEGIN   IF HIGH[I]<=3 THEN K:=3 ELSE K:=HIGH[I];
                CASE MARKS[I] OF
        0:      /*END-NODE : DELETE AND MARK THE FATHER*/
           BEGIN RETAIN[I]:=FALSE;
                MARKS[K]:=MARKS[K]+ORD(INCL[I])
                END;
        1,2,3:  /* PART OF FAN(S) */
                BEGIN   RETAIN[I]:=FALSE;
                        IF ORD(INCL[I])=MARKS[I]
                        THEN MARKS[K]:=MARKS[K]+MARKS[I]
                        ELSE BEGIN MARKS[K]:=4;
                                IF FLAG THEN GOTO 7
                                END
                        END;
        4,5,6:  /*TO BE FIXED */
                BEGIN   RETAIN[I]:=TRUE;
                        MARKS[K]:=4;
                        END
           END;/*OF CASES*/
        END;
        7:   IF FLAG THEN NEWBASE:=K
   END;/*OF PRUNE*/
```

```
PROCEDURE REROOT;/* "REROOTING" THE MOP TO NEWBASE */
VAR     A,B,C,T,SECOND,THIRD: INTEGER;
BEGIN /* 0.INITIALIZE XREF  AND OUT-REP */
        FOR I:=1 TO N DO XREF[I]:=0;
        XREF[NEWBASE]:=1;
        XREF[HIGH[NEWBASE]]:=2;
        XREF[LOW[NEWBASE]]:=3;
        OUTLO[1]:=2; OUTHI[1]:=3;
        OUTLO[2]:=1; OUTHI[2]:=3;
        OUTLO[3]:=1; OUTHI[3]:=2;
     /* 1.MAIN LOOP: CLIMB UP TO THE ROOT RENAMING THE NODES
                AND CREATING THE OUT-REP */
        FATHER:=NEWBASE; K:=4;
        WHILE FATHER>3 DO
        BEGIN   FATHER:=HIGH[FATHER];
                SECOND:=HIGH[FATHER]; THIRD:=LOW[FATHER];
                A:=XREF[FATHER]; B:=XREF[SECOND]; C:=XREF[THIRD];
                IF C>B THEN BEGIN B:=C; C:=SECOND END
                        ELSE C:=THIRD;
                IF B>A THEN BEGIN T:=A; A:=B; B:=T END;
                XREF[C]:=K; OUTLO[K]:=B; OUTHI[K]:=A;
                K:=K+1
                END;
     /* 2.CLEAN-UP: COPY DOWN NOT AFFECTED TRIANGLES */
        FOR I:=FATHER TO N DO WHILE XREF[I]=0 DO
        BEGIN   XREF[I]:=K;
                SECOND:=XREF[HIGH[I]];
                THIRD:=XREF[LOW[I]];
                IF SECOND>THIRD
                   THEN BEGIN OUTHI[K]:=SECOND;OUTLO[K]:=THIRD END
                   ELSE BEGIN OUTHI[K]:=THIRD;OUTLO[K]:=SECOND END;
                K:=K+1
                END
        END;/* OF REROOT */

BEGIN   INCLINATIONS(LOW,HIGH);
        PRUNE(HIGH,TRUE);
        REROOT;
        INCLINATIONS(OUTLO,OUTHI);
        PRUNE(OUTHI,FALSE);
   END
```

```
PROCEDURE MIN_DOM_MOP_EFF ( N: INTEGER;
/*   INPUT: RECURSIVE REPRESENTATION OF THE MOP */
       LOW,HIGH: ARRAY [1..N] OF INTEGER;
/* OUTPUT:VERTICES OF THE DOMINATING MOP */
       VAR RETAIN: ARRAY[1..N] OF BOOLEAN;
       VAR XREF : ARRAY [1..N] OF INTEGER); /* CROSS-REFERENCE */

TYPE  INCLINATION=(NIL,LEFT,RIGHT);
       INTARRAY=ARRAY [1..N] OF INTEGER;
VAR    I,J,K,M,N,FATHER,NEWBASE:INTEGER;
       INCL :ARRAY[4..N] OF INCLINATION;
       OUTLO,OUTHI: INTARRAY; /*OUTPUT REP*/
       FLAG,LASTCASE : BOOLEAN;

PROCEDURE FINDINCLINATIONS;
VAR    /*LOCAL VARIABLES DESCRIBING INCLINATION OF BASE VERTICES*/
       LOLOG,HILOG: ARRAY[2..N] OF INCLINATION;
       L,H: INCLINATION;
   BEGIN LOW[2]:=1;LOLOG[2]:=RIGHT;HILOG[2]:=LEFT;
             LOLOG[3]:=LEFT;HILOG[3]:=RIGHT;
       FOR I:=4 TO N DO
       BEGIN K:=HIGH[I];
             IF LOW[K]=LOW[I] THEN BEGIN L:=LOLOG[K];H:=HILOG[K] END
                              ELSE BEGIN L:=HILOG[K];H:=LOLOG[K] END;
             INCL[I]:=L; LOLOG[I]:=L; HILOG[I]:=H
             END
   END;/* INCLINATIONS */

PROCEDURE DOWNANDOUT; /* PRUNING THE DUAL BINARY  FOREST
                         AND THE REMAINING STUMP */
VAR    PNTR,MARKS : INTARRAY;
       CHANGE : ARRAY [0..7] OF INTEGER;

   PROCEDURE CASES(OLDMARKS,ORDINCL:INTEGER);
   VAR  NEW:INTEGER;
   BEGIN M:=MARKS[I]; NEW:=10*OLDMARKS + ORDINCL;
        IF M=0          THEN M:=NEW
   ELSE IF(M=1)OR(M=2)  THEN IF ORDINCL=M THEN M:=NEW+2
                                          ELSE M:=NEW+4
   ELSE IF(M=3)OR(M=4)  THEN IF ORDINCL+2=M THEN M:=NEW+2
                                          ELSE M:=NEW+4
   ELSE IF(M=12)OR(M=14)OR
          (M=21)OR(M=23)OR
          (M=32)OR(M=34)OR
          (M=41)OR(M=43)  THEN M:=7
                          ELSE LASTCASE:=TRUE;
        MARKS[K]:=M
        END; /* OF CASES */

   PROCEDURE DOWN;
   BEGIN RETAIN[I]:=TRUE;
        IF FLAG THEN MARKS[K]:=7 /* NOT THE FIRST TO BE FIXED */
                ELSE BEGIN NEWBASE:=I; J:=I; FLAG:=TRUE;
                        RETAIN[LOW[I]]:=TRUE;
                        RETAIN[HIGH[I]]:=TRUE;
                        WHILE K>3 DO
                        BEGIN PNTR[K]:=J; J:=K; K:=HIGH[K] END;
                        PNTR[3]:=J; J:=K + LOW[J] -2
                        END
        END/* OF THE LAST CASE GOING DOWN */;
```

```
BEGIN   FLAG:=FALSE; LASTCASE:=FALSE;
        CHANGE[0]:=0; CHANGE[1]:=2; CHANGE[3]:=6; CHANGE[5]:=4;
        CHANGE[7]:=7; CHANGE[2]:=1; CHANGE[6]:=3; CHANGE[4]:=5;
        FOR I:=1 TO N DO
        BEGIN MARKS[I]:=0; PNTR[I]:=0; RETAIN[I]:=FALSE END;
        FOR I:=N DOWNTO 4 DO IF PNTR[I]=0 /*NOT ON THE TRUNK*/
        THEN BEGIN K:=HIGH[I]; IF K<=3 THEN /* ENCODE THE INCOMING
                SIDE OF THE BASE TRIANGLE */ K:=K+LOW[I]-2;
                CASES(MARKS[K],ORD(INCL[I]));
                IF LASTCASE THEN BEGIN DOWN; LASTCASE:=FALSE END;

                END/* OF SCANNING DOWN */;
/*   NOW ALL THE INFORMATION NEEDED IS IN FIELDS 'MARKS' AND
     'PNTR' OF TRUNK NODES. MARKS[3] HAS TO BE PREPARED
     AND THE PRUNING OF THE TRUNK MAY START */

        IF FLAG THEN /* A NEWBASE HAD BEEN DISCOVERED */
        BEGIN CASE J OF
        1:      IF MARKS[3]=0 THEN M:=MARKS[2]
                        ELSE M:=10*MARKS[3] + MARKS[2];
        2:      IF MARKS[1]=0THEN M:=CHANGE[MARKS[3]]
                        ELSE M:=10*MARKS[1] + CHANGE[MARKS[3]];
        3:      IF MARKS[2]=0 THEN M:=MARKS[1]
                        ELSE M:=10*CHANGE[MARKS[2]] + MARKS[1]
              END; /* OF CASES */
                MARKS[3]:=M;
                I:=3; K:=PNTR[3]; J:=4-J /* THE 'DANGLING' VERTEX */;
                WHILE K<NEWBASE DO IF FLAG
                THEN /* NO TRUNK TRIANGLE FIXED YET */
                BEGIN CASES(CHANGE[MARKS[K]],3-ORD(INCL[PNTR[K]]));
                        IF LASTCASE THEN
                        BEGIN LASTCASE:=FALSE;
                                FLAG:=FALSE;
                                RETAIN[J]:=TRUE
                                END;
                        I:=K; K:=PNTR[I];
                        IF (LOW[I]=LOW[K]) OR (LOW[I]=HIGH[K])
                            THEN J:=HIGH[I] ELSE J:=LOW[I]
                        END
                ELSE /*FIX THE REST OF TRIANGLES IN TRUNK */
                BEGIN  RETAIN[HIGH[I]]:=TRUE;
                        RETAIN[LOW[I]]:=TRUE;
                        K:=NEWBASE;
                        WHILE I<NEWBASE DO
                        BEGIN RETAIN[I]:=TRUE ;
                                I:=PNTR[I] END
                      END;/* OF FIXING THE TRUNK */
                IF I<NEWBASE THEN
                BEGIN CASES(0,0);
                        IF LASTCASE THEN RETAIN[J]:=TRUE
                        END
                END /* OF GOING UP THE TRUNK */
        ELSE FOR I:=1 TO 3 DO RETAIN[I]:=TRUE /*THE OLD BASE WILL DO */
        END; /* OF DOWN AND OUT */

BEGIN   FINDINCLINATIONS;
        DOWNANDOUT
        END
```