ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XVIII, 1 (1983), p. 151-157



Z. PALKA (Poznań)

# A NEW ALGORITHM CALCULATING THE DISTANCE BETWEEN BINARY ARBORESCENCES

**1. Procedure declaration.** An *arborescence* $A$ is defined as a directed tree that has a root, i.e., a tree with the vertex $v_0$, such that all vertices of the tree can be reached by a path starting from $v_0$. Let $X = \{1, 2, \ldots, n\}$ be the set of *pendant* (*terminal*) *vertices* of an arborescence $A$, i.e., $d^-(i) = 1$ and $d^+(i) = 0$ for $i = 1, 2, \ldots, n$, where $d^-(i)$ and $d^+(i)$ denote the inner demi-degree and the outer demi-degree of the vertex $i$, respectively. By a *binary arborescence* we mean such an arborescence in which $d^+(v) = 2$ for each $v \notin X$. Since in this paper we consider only binary arborescences, the word "binary" is omitted.

The procedure *arbmetric* calculates the distance between arborescences (more precisely, the distance between hypergraphs generated by arborescences) with the same set of pendant vertices, proposed in [1], and is a modification of the procedure *hypergraphmetrics* presented in [2].

The arborescence $A$ generates the hypergraph $H_A = (X, \mathscr{E}_A)$, where the class of edges $\mathscr{E}_A$ is defined as follows: each non-pendant vertex $v$ generates exactly one edge in $\mathscr{E}_A$ which consists of those elements of the set $X$ which are the pendant vertices of the subarborescence generated by the vertex $v$. The distance between $H_{A_1} = (X, \mathscr{E}_{A_1})$ and $H_{A_2} = (X, \mathscr{E}_{A_2})$ was defined in [1] as

$$\varrho(H_{A_1}, H_{A_2}) = \frac{1}{n-1} \min_{p \in \mathscr{P}} \sum_{i=1}^{n-1} \frac{|E_i \Delta F_{p(i)}|}{|E_i \cup F_{p(i)}|},$$

where $p(i)$ is the $i$-th element of the permutation $p$ of the first $n-1$ integers, $\mathscr{P}$ is the set of all such permutations, and $E_i \in \mathscr{E}_{A_1}$, $F_{p(i)} \in \mathscr{E}_{A_2}$, $i = 1, 2, \ldots, n-1$.

Now, let us describe a new element of the algorithm given in this paper, which is a better representation of a hypergraph by some vector. First, consider the example of the arborescence presented in Fig. 1.

This arborescence can be represented by the hypergraph $H_A = (X, \mathscr{E}_A)$ where $X = \{1, 2, \ldots, 9\}$ and the class of edges $\mathscr{E}_A$ may be the following:

$$E_1 = \{4, 5\}, \quad E_2 = \{2, 7\}, \quad E_3 = \{8, 9\}, \quad E_4 = E_1 \cup \{1\},$$

$$E_5 = E_2 \cup \{6\}, \quad E_6 = E_5 \cup \{3\}, \quad E_7 = E_3 \cup E_6, \quad E_8 = E_4 \cup E_7.$$

Now, note that we are always able to form the class $\mathscr{E}_A$ in such a way that

(a) $|E_1| = 2$, $|E_i| \leqslant |E_{i+1}|$ $(i = 1, 2, \ldots, n-2)$,

(b) each successive edge can be either

1° the set of two different vertices, or

2° the sum of the edge already defined and of a new vertex, or

3° the sum of two edges already defined.

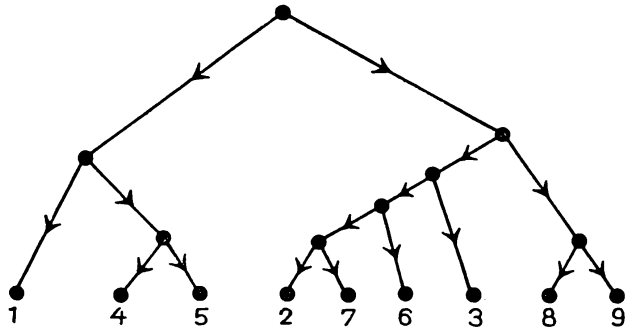


Fig. 1

For a hypergraph generated by a binary arborescence, no other situation is possible. Thus, we can represent each of the $n-1$ edges of $H_A$ by two numbers only, which are either

1° numbers of vertices, or

2° the index (with a minus sign) of the edge already defined and the number of a new vertex, or

3° two indices (with a minus sign) of the edges already defined.

The components of the vector representing the arborescence given in Fig. 1 are the following (for clarity we write each edge in parentheses):

(4, 5), (2, 7), (8, 9), (−1, 1), (−2, 6), (−5, 3), (−3, −6), (−4, −7).

Therefore, we need only $2n-2$ elements as components of the vector representing an arborescence, whereas the number of components needed in the previous algorithm [2] was much greater for large arborescences.

Data:

$m$ — the number of pendant vertices minus 1, i.e., $m = n-1$;

$A1[1 : 2m]$,

$A2[1 : 2m]$ — arrays representing the first and the second arborescences, respectively;

```
procedure arbmetric(m,A1,A2,d,fac,inf,assign);
 value m,fac,inf;
 integer m,fac,inf;
 real d;
 integer array A1,A2;
 procedure assign;
 begin
  integer b1,b2,i,j,k,k1,k2,l,l1,l2,s,w;
  integer array B1,B2[1:m];
  procedure size(m,l,A,B);
   value m;
   integer m,l;
   integer array A,B;
   begin
    integer a,b,i,j,k;
    B[1]:=l:=2;
    j:=3;
    for i:=2 step 1 until m do
     begin
      a:=A[j];
      b:=A[j+1];
      if a>0
       then
       begin
        B[i]:=2;
        l:=l+2;
        go to ET
       end a>0
       else
        if b>0
```

```
            then k:=B[abs(a)]+1
            else k:=B[abs(a)]+B[abs(b)];
          B[i]:=k;
          l:=l+k;
ET:       j:=j+2
        end i
      end size;
      size(m,l1,A1,B1);
      size(m,l2,A2,B2);
      begin
       integer array C1[1:l1],C2[1:l2],D[1:m,1:m],E[1:m];
       procedure table(m,A,B,C);
        value m;
        integer m;
        integer array A,B,C;
        begin
         integer a,b,e,i,j,k,l;
         integer array F[1:m];
         j:=k:=1;
         for i:=1 step 1 until m do
          begin
           F[i]:=k;
           a:=A[j];
           b:=A[j+1];
           if a>0
            then
            begin
             C[k]:=a;
             C[k+1]:=b;
             k:=k+2
```

```
end a>0
else
begin
 s:=abs(a);
 e:=F[a];
 a:=k+B[a]-1;
 for l:=k step 1 until a do
  begin
   C[l]:=C[e];
   e:=e+1
  end l;
 if b>0
  then
  begin
   C[a+1]:=b;
   k:=a+2
  end b>0
  else
  begin
   b:=abs(b);
   e:=F[b];
   b:=a+B[b];
   for l:=a+1 step 1 until b do
    begin
     C[l]:=C[e];
     e:=e+1
    end l;
   k:=b+1
  end b<0
end a<0;
```

```
      j:=j+2
     end i
    end table;
   table(m,A1,B1,C1);
   table(m,A2,B2,C2);
   k1:=0;
   for i:=1 step 1 until m do
    begin
     l1:=k1+1;
     b1:=B1[i];
     k1:=k1+b1;
     k2:=0;
     for j:=1 step 1 until m do
      begin
       w:=0;
       l2:=k2+1;
       b2:=B2[j];
       k2:=k2+b2;
       for l:=l1 step 1 until k1 do
        begin
         s:=C1[l];
         for k:=l2 step 1 until k2 do
          if s=C2[k]
           then
           begin
            w:=w+2;
            go to END
           end k,s=C2[k];
END:    end l;
      s:=b1+b2;
```

```
        D[i,j]:=(s-w)/(s-w+2)×fac
      end j
    end i;
    assign(m,D,E,s,inf);
    d:=s/(m×fac)
  end
end arbmetric
```

*fac* — the factor by which the cost matrix $c$ (see the procedure *assign*) must be multiplied in order to obtain the integer coefficients of $c$;

*inf* — the maximal allowed number of type **integer** in the computer;

*assign* — the procedure with the following procedure head:

**procedure** *assign*(*m*, *c*, *p*, *total*, *inf*);
**value** *m*, *inf*;
**integer** *m*, *total*, *inf*;
**integer array** *c*, *p*;

this procedure is published in [3] and finds $\min \sum_{i=1}^{m} c_{ip_i}$, where $(p_1, p_2, \ldots, p_m)$ is the permutation of the numbers $(1, 2, \ldots, m)$.

Result:

*d* — the distance between arborescences.

**2. Method used.** The method is similar to that in [2].

**3. Certification.** The procedure has been verified on the Odra 1204 computer for the problems described in [1].

**References**

[1] M. Karoński and Z. Palka, *On Marczewski-Steinhaus type distance between hypergraphs*, Zastos. Mat. 16 (1977), p. 47-57.

[2] — *Algorithm 69: Two distances between hypergraphs*, ibidem 16 (1980), p. 671-679.

[3] J. Kucharczyk and M. Sysło, *Algorytmy optymalizacji w języku ALGOL 60*, Warszawa 1975.

MATHEMATICAL INSTITUTE
A. MICKIEWICZ UNIVERSITY
60-769 POZNAŃ

*Received on 28. 3. 1980*