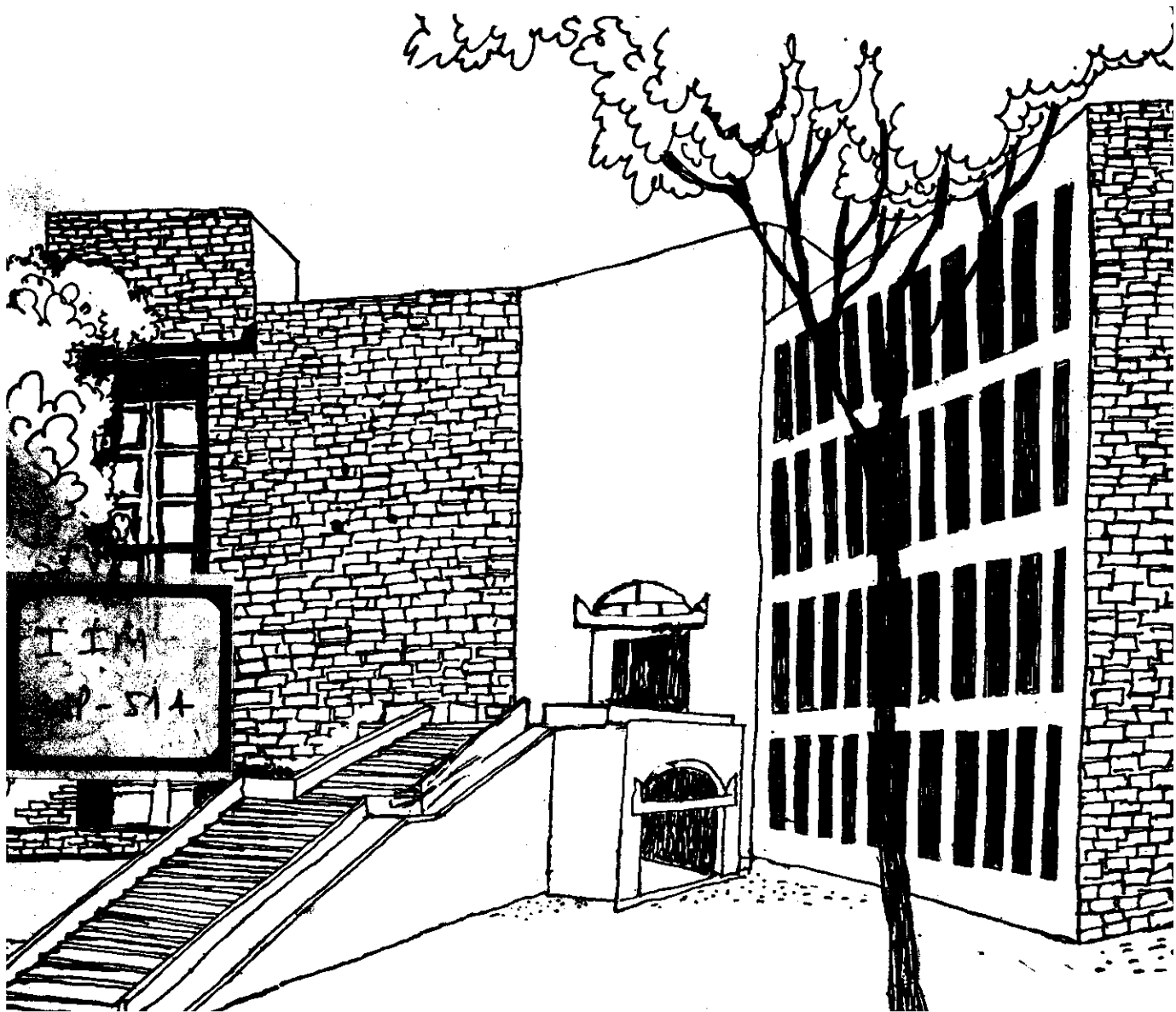




Working Paper




THE MINIMUM WEIGHT ROOTED ARBORESCENCE
PROBLEM : A BRANCH AND BOUND SOLUTION

By

V. Venkata Rao

&

L.F. Mc Ginnis

WP514

WP
1984
(514)

W P No. 514

June 1984

The main objective of the working paper series of the IIMA is to help faculty members to test out their research findings at the pre-publication stage.

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD-380015
INDIA

THE MINIMUM WEIGHT ROOTED ARBORESCENCE PROBLEM:
A BRANCH AND BOUND SOLUTION

V. Venkata Rao* and L.F. Mc Ginnis**

I. INTRODUCTION

In this paper we develop a branch and bound technique for the minimum weight rooted arborescence problem in an acyclic graph with weights on nodes. This problem was not studied in the literature before. The importance of the problem lies in the fact that the uncapacitated plant location problem is a special case of this problem; also, the problem was encountered by the authors as a sub-problem in a decomposition strategy for the optimal lot sizing problem in multi-stage production systems [4]. It is to be emphasised here that the arborescence we deal with need not be a spanning arborescence, and hence our problem is different from the minimum weight rooted spanning arborescence problem [3].

II. TERMINOLOGY AND NOTATION

We deal with graphs which satisfy the following properties: (1) Each arc of the graph is directed, (2) There are no directed cycles embedded in the graph, (3) The graph is connected; one of the vertices which does not have any incoming arcs, and which has a connected path to every other node in the graph is specified as the root of the graph, (4) Associated with every vertex J , there is a weight a_j , which can be positive, negative, or zero,

* Indian Institute of Management, Ahmedabad.

** School of Industrial and Systems Engineering, Georgia Institute of Technology.

(5) Arcs do not have any weights. In our future discussion we refer to a graph such as the above by the symbol G .

G consists of N nodes, $N \geq 0$; they are indexed with consecutive integers $1, \dots, N$ in the order exactly opposite to the topological order; that is, each arc in the graph is directed from a higher numbered node to a lower numbered node. If an arc is directed from node i to node J , i is said to be an immediate predecessor of J , and J an immediate successor of i . According to this terminology the root does not have any predecessors and bears the index N . The set $P(J)$ consists of the indices of the immediate predecessors of J , and $S(J)$ the indices of the immediate successors of J . F and L are sets consisting of vertex indices such that $J \in F \rightarrow P(J) = \emptyset$, and $J \in L \rightarrow S(J) = \emptyset$. According to this notation $F = \{ N \}$.

A subgraph S of G is called a rooted arborescence (RA) if

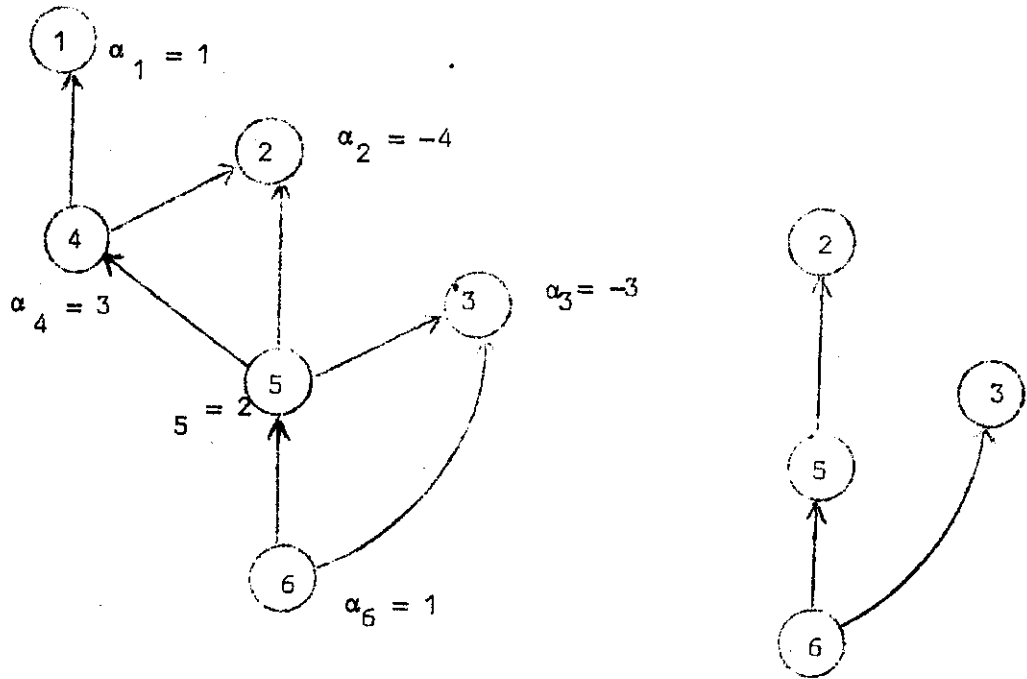
- (1) S contains the root as one of its vertices.
- (2) S is connected, and (3) No two arcs of S are directed towards the same vertex.

The sum of weights of the vertices in the arborescence is called the weight of the arborescence.

A rooted path in G is a sequence of nodes and arcs which can be written as $(N, (N, J), J, (J, K), \dots, (L, I), I)$. The first vertex in a rooted path must be the root. The sum of weights of the vertices in a rooted path is called the weight of the path.

The minimum weight rooted arborescence (MRA) problem in an acyclic graph G with weights on nodes is to find a rooted arborescence in G that has a weight less than or equal to the weight of any other rooted arborescence in G .

Figure-1 gives an example of MRA in an acyclic graph.



(a) Rooted acyclic graph G.

(b) MRA of G.

Weight of MRA = -4.

Figure-1. Illustration of MRA.

Mathematically the problem can be stated as:

MRA Problem

$$\text{Min } Y \in \{0,1\} \left[\begin{array}{l} \sum_{j=1}^N \alpha_j Y_j \\ Y_j \leq \sum_{i \in P(j)} Y_i, \quad j = 1, \dots, N-1, Y_N = 1 \end{array} \right] \quad (1)$$

In the above formulation Y_j 's are zero-one variables which indicate the presence ($Y_j = 1$) or absence ($Y_j = 0$) of each node in the selected arborescence. The above formulation is concerned only with nodes. This is because the arcs of the graph do not have any weights; therefore, once we

know which vertices are to be included in the arborescence, selection of arcs for the arborescence is simple.

III. RELATIONSHIP BETWEEN MRA AND THE UNCAPACITATED FACILITY LOCATION PROBLEM.

The uncapacitated facility location problem (UL) [1], which is an NP-Complete problem, can be formulated as an MRA problem. Since UL is a special case of MRA problem, it is unlikely that an efficient algorithm exists for MRA problem either. Problem UL can be stated verbally as below: Suppose there are n demand points each of which has a non-zero demand d_j , $j = 1, \dots, n$ for a particular commodity. Suppose the demands can be met by establishing facilities at some or all of m given sites (or sources). Location of a facility at site i will incur a fixed cost f_i ; to meet a unit of demand from source i to destination J will involve a cost C_{ij} . There are no capacity restrictions at any of the facilities. The problem is to determine the sites at which facilities are to be located and the quantity of demand of market J to be met by the facility at source i such that the total fixed and variable costs are minimised. Because there is no limit on the capacity of any facility, there exists an optimal solution for UL such that each destination is supplied by not more than one source.

The problems MRA and UL can be shown to be related by first proposing an acyclic graph $G(UL)$ to represent UL. Assign m nodes, one for each site, and let the weight of each of the nodes be the corresponding fixed cost of locating a facility. Assign mn nodes, called trans-shipment nodes, one for each source-destination pair. Let the weight of each of these nodes be the total variable cost of meeting all the demand for the corresponding destination

from the corresponding source. Further, assign one node for each destination and let the weight of each of these nodes be a large negative number such that its absolute value is greater than the sum of weights of any pair of source and trans-shipment nodes. In addition, let there be a root node with weight zero. The root is connected with every source node by an arc directed away from the root. Each source node is connected with its corresponding trans-shipment nodes by arcs directed towards the trans-shipment nodes. Similarly each trans-shipment node is connected to its corresponding destination-node by means of an arc directed towards the latter node.

It can be easily shown that an optimal solution of UL, in which each demand point J is supplied by only one source i , can be represented as a minimum weight rooted arborescence in graph $G(UL)$. Similarly it can be shown that given an MRA of $G(UL)$ an optimal solution to UL can be easily constructed [4]

IV. LINEAR RELAXATION OF MRA-PROBLEM AND ITS DUAL.

In the branch bound method to be developed later in this paper, at each node of the branch and bound tree we have to deal with a restricted version of MRA problem, which will be referred to as MRA_{θ} . In MRA_{θ} , we have to find a minimum weight rooted arborescence among all the rooted arborescences which include certain vertices, specified by a set θ_{in} , and exclude certain other vertices, specified by a set θ_{out} ; the remaining nodes, free to be included or excluded are contained in a third set θ_{free} .

At each node, in addition to MRA_{θ} , we will be interested in two related problems: (1) the linear relaxation, PRA_{θ} , of the problem MRA_{θ} , and (2) the dual, DRA_{θ} , of the linear program PRA_{θ} . The three problems

are stated below in a form convenient for algorithmic development.

$$\underline{MRA}_\theta \quad \cdot \quad \text{Min} \quad \sum_{J=1}^N \alpha_J Y_J \quad (2)$$

$$\text{s.t.} \quad Y_J \leq \sum_{i \in P(J)} Y_i \quad J = 1, \dots, N-1 \quad (3)$$

$$Y_J = 1 \quad J \in \theta_{in} \quad (4)$$

$$Y_J = 0 \quad J \in \theta_{out} \quad (5)$$

$$Y_J \in \{0,1\} \quad J \in \theta_{free} \quad (6)$$

$$\underline{PRA}_\theta \quad \text{Min} \quad \sum_{J \in \theta_{in} \cup \theta_{out}} \alpha_J Y_J \quad (7)$$

$$\text{s.t.} \quad -Y_J + \sum_{i \in P(J)} Y_i \geq 0 \quad J \notin F \quad (8)$$

$$-Y_J = -1 \quad J \in \theta_{in} \quad (9)$$

$$-Y_J = 0 \quad J \in \theta_{out} \quad (10)$$

$$-Y_J \geq -1 \quad J \in \theta_{free} \quad (11)$$

$$Y_J \geq 0 \quad \text{all } J \quad (12)$$

By associating the dual variables U_J with (8) and V_J with (9), (10), and (11), we can write DRA_θ as

$$\underline{\text{DRA}}_{\theta} \quad \text{Find} \quad - \text{Min} \quad \sum_{J \in \theta_{\text{in}} \cup \theta_{\text{free}}} V_J \quad (13)$$

$$\text{s.t.} \quad U_J + V_J \geq -\alpha_J \quad J \in L \quad (14)$$

$$U_J + V_J \geq -\alpha_J + \sum_{i \in S(J)} U_i \quad J \notin L, J \notin F \quad (15)$$

$$V_J \geq -\alpha_J + \sum_{i \in S(J)} U_i \quad J \in F \quad (16)$$

$$U_J \geq 0 \quad J \notin F \quad (17)$$

$$V_J \geq 0 \quad J \in \theta_{\text{free}} \quad (18)$$

If all J's are in θ_{free} then PRA_{θ} will be referred to as PRA and DRA_{θ} as DRA.

Before presenting the branch and bound scheme for the MRA problem, let us present solution algorithms for MRA_{θ} and DRA_{θ} , because these will be used in the branch and bound scheme.

V. A HEURISTIC FOR MRA_{θ} .

The heuristic constructs a rooted arborescence which includes and/or excludes any desired nodes, and has a low sum of node weights.

A key idea on which the heuristic is based is this: in a rooted arborescence, every node is connected to the root by a directed path; hence,

we can look upon a rooted arborescence as a union of rooted paths. Therefore, for our purposes, we would like to choose a collection of rooted paths whose union is an arborescence and which has a low sum of node weights. In choosing the rooted paths, we will be concerned with selection of nodes only; if any such selected node has more than one arc incident on it, we can drop all but one such arc to get the rooted arborescence.

The heuristic consists of three phases. Initially, we will remove from the graph all the nodes prohibited from appearing in the arborescence; along with those nodes all the arcs incident on them will also be removed.

We will set two buckets B_1 and B_2 and keep filling them with node indices as the algorithm progresses. When certain conditions are satisfied the indices from B_2 will be transferred to B_1 ; under some other conditions, some indices in B_1 will be erased. Finally, when the algorithm ends, the indices in B_1 correspond to the nodes in the selected arborescence.

Phase 1. For each node in the graph we find the minimum weight path from the root. If the minimum of the minimum weight paths found above has a negative weight, then we place the indices of all the nodes in that path in bucket B_1 . Then the weights of selected nodes are updated to zero. Once again, in the updated graph the rooted path with minimum weight is found, and if its weight is negative, then all its node indices are entered in B_1 , while at the same time updating the weights of its nodes to zero. We continue this process until the minimum weight rooted path in the graph happens to have a non-negative weight.

We can stop here considering the nodes in B_1 as constituting the rooted arborescence; but such a strategy might lead to some obvious omissions and

hence might result in poor solutions. Phase-2 helps to avoid such omissions.

Phase-2. As long as there are negatively weighted nodes in the graph we continue to select the minimum weight rooted path and update the weights of its nodes to zero. But, this time, we place the indices of the selected nodes in bucket B_2 . Also, we keep a running total of the original weights of nodes in B_2 ; when this total becomes negative all the indices in B_2 are transferred to B_1 . This phase would end only when no negative-weight nodes are left in the graph.

Phase-3. There is a chance that even after the above two phases some vertices required to be present in the rooted arborescence are not selected for the arborescence. We have to force in such left out nodes into the arborescence. To accomplish this, we consider each node in the graph and whenever we encounter a node which is in θ_{in} , and not in B_1 , then we select the minimum weight path from root to the node and enter the indices of the nodes on the path into B_1 .

At the end of phase-3 the indices in B_1 correspond to the vertices of the desired rooted arborescence.

Example-1. Suppose we want to develop an MRA₀ for the graph given in Figure-2, using the algorithm described above. Let $\theta_{in} = \{4, 5\}$ and $\theta_{out} = \{3\}$.

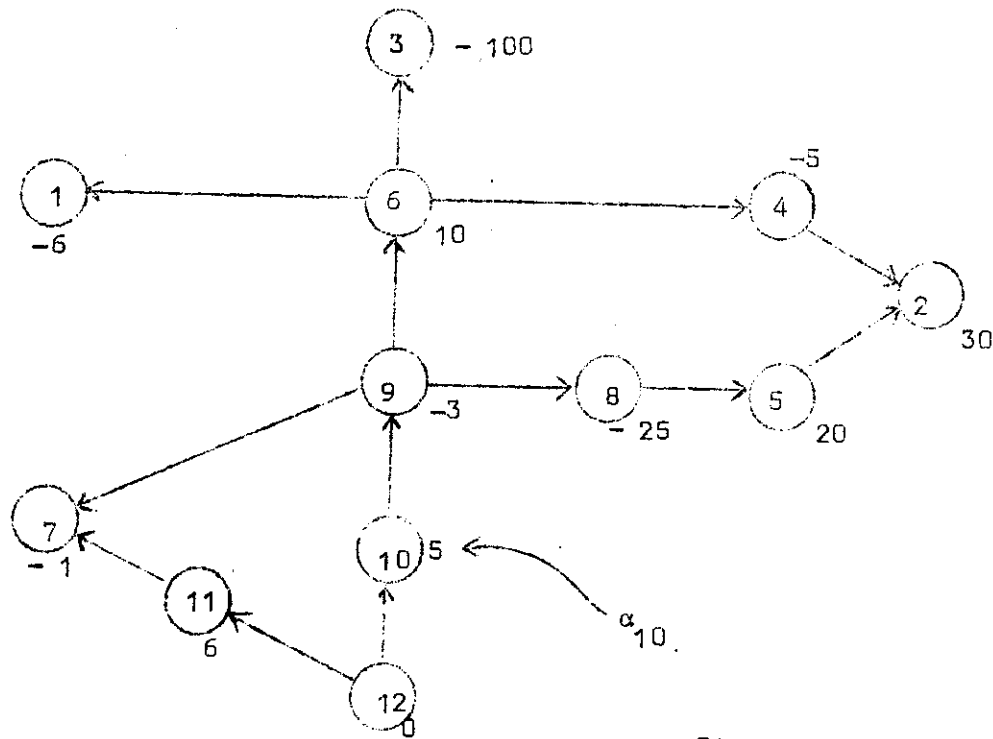


Figure-2

We have to consider the graph, after removing node 3. In phase-1, the node with the minimum weight rooted path turns out to be node 8, and the corresponding path is 12-10-9-8 with the weight -23. Therefore, B_1 is filled with indices 12, 10, 9, 8 and the weights of these nodes are updated to zero. In the updated graph, node 3 has the minimum weight rooted path, which is 12-10-9-7 with the weight -1. Therefore, node 7 gets added to B_1 , and its weight is updated to zero. In the updated graph at this point there is no node with a minimum weight rooted path of negative weight; but, there are some nodes with negative weights. Therefore, we begin phase 2. The nodes that are entered first in B_2 are 1 and 6, because at this stage the minimum of the minimum weight rooted paths is 12-10-9-6-1, of which the nodes 12, 10 and 9 are already selected for B_1 . At this stage the sum of the original weights of B_2 's contents is $-6+10 = 4$. In the updated graph, vertex 4 has the shortest

path from the root. Therefore, 4 is entered in B_2 , and the weight of vertex 4 is updated to zero. The contents of B_2 at this stage are 1, 4 and 6 and the sum of their original weights is $-6-5+10 = -1$; since this sum is negative, the indices 1, 4 and 6 are removed from B_2 and entered in B_1 . In the graph with updated weights there are no negative weight nodes. Hence, phase-2 ends. At this point the vertices in B_1 are 1, 4, 6, 7, 9, 8, 10, 12. Since the nodes 2 and 5 which are the members of Θ_{in} are not in B_1 , we have to execute phase-3 which brings 2 and 5 into B_1 . Thus, the selected arborescence, shown in Figure-3, consists of nodes 1, 2, 4, 5, 6, 7, 8, 9, 10, and 12.

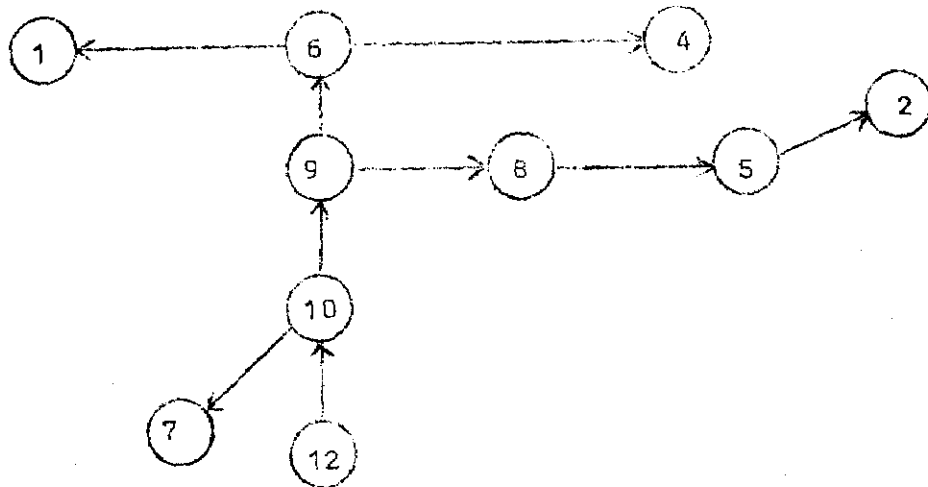


Figure-3

VI. A HEURISTIC FOR DRA_{θ} .

DRA_{θ} is a linear program. We propose here a heuristic that constructs a good feasible solution to DRA_{θ} . The reason for using a heuristic instead of the simplex method is that the heuristic can exploit the special structure

of the problem and hence might consume less storage space and computational time.

The Heuristic exploits the following useful features of the constraints of DRA_{θ} :

1. There is a one-to-one correspondence between the vertices of G and the constraints of DRA_{θ} .
2. Each node J of G , except the root, has two variables U_J and V_J associated with it; the root, N , has only the variable V , but does not have U 's.
3. The variables that appear in the constraint for a node J are: U_J , V_J and the U 's corresponding to all immediate successors of J . Thus, U 's serve as the link-variables between different constraints. The V 's on the other hand do not repeat themselves. Figure-4 illustrates the above features pictorially.

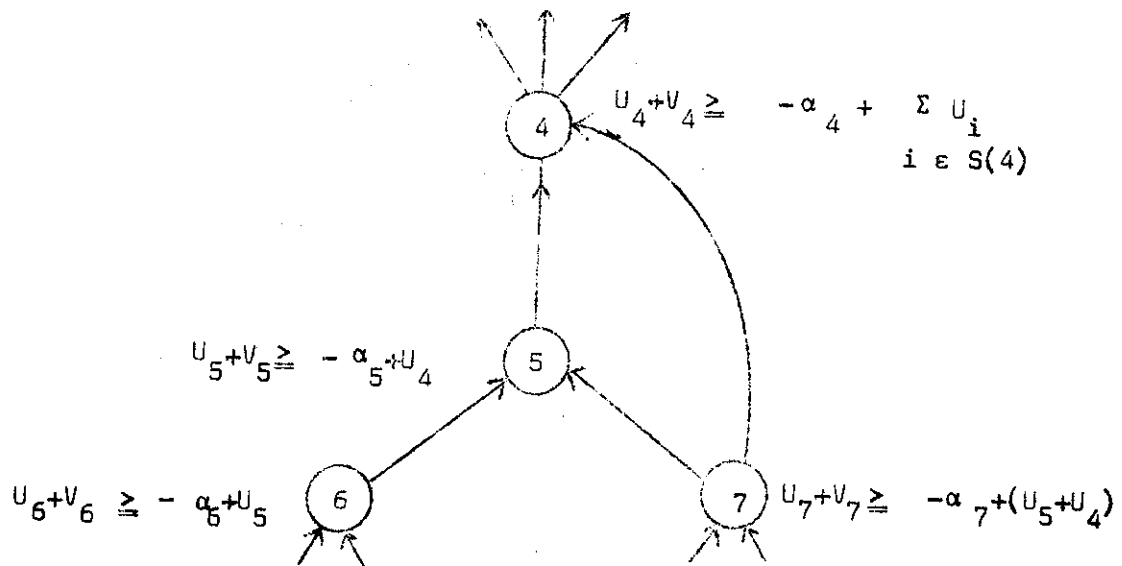


Figure-4. Constraints in DRA_{θ} : A Typical Situation.

The above features suggest a useful guideline for our heuristic. Suppose we arrive at a feasible solution somehow, and then want to improve it. In the process of modification, if we were to change the value of U_J at a node J then the change would get transmitted to all the predecessors of J , because the values of the right hand sides of these constraints will change. Thus a change in U_J is likely to ripple down the graph necessitating further changes at the predecessors of J . A change in V_J , on the other hand, by itself does not disturb the variables at other nodes, unless it is accompanied by a change in U_J .

In the heuristic to be described below, which is a multi-pass heuristic, we first find a feasible solution, and then try to improve it by making several passes through the graph.

In constructing the initial feasible solution, we consider nodes in the increasing order of node index K , assigning values to U_K and V_K according to the decision table given in Table-1. The major idea behind the given rules is that initially, the surplus, that is the value of left hand side minus the value of right hand side, of each constraint should be as small as possible, and that at the same time the value of U should be as large a non-negative value as possible. Such a procedure is likely to yield a feasible solution with a large value of V_N and would be particularly amenable to improvement by the strategies to be described later.

After the initialisation, we will attempt to improve the solution. The improvement procedure consists of cycling through the graph, visiting each node in the decreasing order of node index in each cycle, and attempting to change the value of V corresponding to the node being visited. The type of

change being attempted is not the same in consecutive cycles. There are two types of changes, type-A and type-B. We alternate between these two changes from cycle to cycle. The rules in the two types of changes are so designed that the solutions before and after a change are feasible, and the value of the objective function after a change is not worse than that before the change. We keep executing one cycle after another till we happen to execute one which cannot bring about any change in the solution. Another important aspect of the changes is that when we change the variables at the node under consideration, the changes do not get transmitted beyond its immediate predecessors.

The difference between the two types of changes is as below:

In type-A change we attempt to improve the solution by decreasing the value of V at the node under consideration; whereas, in type-B, we attempt to bring about an improvement by increasing the value of V at the node under consideration.

Figure-5 gives an algorithmic description of the heuristic.


```

Begin
  FOR J = 1 TO N      DO
    Find  $U_J, V_J$  using Table-1
  ENDFOR

  Type of change in next cycle  $\leftarrow$  Type-A.

  REPEAT

    CASE   Type of change OF

    TYPE-A :  $X \leftarrow 0$ 

              FOR J = N-1 DOWNTQ 1      DO
                Change Variables at J and all K,  $K \in P(J)$ 
                  using the algorithm in Figure-6.
              ENDFOR

              IF the solution did not change in the above
                pass THEN  $X \leftarrow 1$   ENDIF

              Type of change  $\leftarrow$  Type-B

    TYPE-B :  $X \leftarrow 0$ ;

              FOR J = N DOWNTQ 1      DO
                Change variables at J, and at all K  $\in P(J)$ 
                  using the algorithm in Figure-7.
              ENDFOR

              IF the variables did not change in the above
                pass THEN  $X \leftarrow 1$   ENDIF

              Type of change  $\leftarrow$  Type-A

    ENDCASE

  UNTIL  $X = 1$ 

END.

```

Figure-5. Algorithm to find a feasible solution to DRA.

$K \in F$											
$K \in L$	✓	✓	✓	✓	✓						
$K \notin F$ and $K \notin L$						✓	✓				
$K \in \theta_{in}$	✓	✓				✓	✓				
$K \in \theta_{out}$			✓							✓	
$K \in \theta_{free}$				✓	✓			✓	✓		
$a_K \leq 0$	✓			✓							
$a_K > 0$		✓			✓						
$-a_K + \sum_{i \in S(K)} \epsilon U_i \geq 0$						✓		✓			
$-a_K + \sum_{i \in S(K)} \epsilon U_i < 0$							✓		✓		
U_K	$-a_K$	0	0	$-a_K$	0	$-\sum_{i \in S(K)} \epsilon U_i$	0	$-\sum_{i \in S(K)} \epsilon U_i$	0	0	-
V_K	0	$-a_K$	$-a_K$	0	0	0	$-\sum_{i \in S(K)} \epsilon U_i$	0	0	$-\sum_{i \in S(K)} \epsilon U_i$	$-\sum_{i \in S(K)} \epsilon U_i$

Table 1. Decision Tables for finding the values U_K and V_K for a node, K , in the initial solution to DRA_{θ} .

```

BEGIN

    Δ ← 0

    For each K ∈ P(J) DO

        Perform one of the following 3 cases, whichever is
        applicable.

        (i) K ∈ Θin : Δ ← ∞
        (ii) K ∈ Θout : Δ ← Δ
        (iii) K ∈ Θfree : Δ ← max { Δ, VK }

    ENDFOR

    Δ ← Min { UJ, Δ }

    VJ ← VJ + Δ
    UJ ← UJ - Δ

    For each K ∈ P(J) DO

        IF K ∈ Θfree THEN VK ← Max { VK - Δ, 0 }

        ELSE VK ← VK - Δ

        ENDIF

    ENDFOR

END

```

Figure-6. Algorithm to apply type-A change at a node J.

(Surplus (J) = (Left hand side - Right hand side) of the constraint for J.)

BEGIN

IF SURPLUS (J) > 0 THEN

Perform one of the following 3 cases, whichever is applicable.

(i) $J \in \theta_{\text{free}}$: $\Delta \leftarrow \text{Min} \{ V_J, \text{SURPLUS}(J) \}$

(ii) $J \in \theta_{\text{in}}$: $\Delta \leftarrow \text{SURPLUS}(J)$

(iii) $J \in \theta_{\text{out}}$: $\Delta \leftarrow 0$

$V_J \leftarrow V_J - \Delta$

$U_J \leftarrow U_J$

ELSE

FOR each $K \in P(J)$ DO

Perform one of the following 5 cases whichever is applicable.

(i) $K \in \theta_{\text{in}}$ and $\text{SURPLUS}(K) > 0$: $\Delta \leftarrow \text{Min} \{ \Delta, \text{SURPLUS}(K) \}$

(ii) $K \in \theta_{\text{in}}$ and $\text{SURPLUS}(K) = 0$: $\Delta \leftarrow 0$

(iii) $K \in \theta_{\text{out}}$: $\Delta \leftarrow \Delta$

(iv) $K \in \theta_{\text{free}}$ and $\text{SURPLUS}(K) > 0$: $\Delta \leftarrow \text{Min} \{ \Delta, \text{SURPLUS}(K) \}$

(v) $K \in \theta_{\text{free}}$ and $\text{SURPLUS}(K) = 0$: $\Delta \leftarrow 0$

ENDFOR

IF $J \in \theta_{\text{in}}$ THEN $\Delta \leftarrow \Delta$

ELSE IF $J \in \theta_{\text{out}}$ $\Delta \leftarrow 0$ ELSE $\Delta \leftarrow \text{MIN} \{ \Delta, V_J \}$ ENDIF

contd.

Figure-7

```

ENDIF
IF  $\Delta > 0$  THEN
     $V_J \leftarrow V_J - \Delta$  ;  $U_J \leftarrow U_J + \Delta$ 
    FOR each  $K \in P(J)$  DO
        IF  $K \in \Theta_{out}$  THEN  $V_K \leftarrow V_K + \text{Max}\{\Delta - \text{SURPLUS}(K), 0\}$ 
    ENDFOR
ENDIF
ENDIF
END

```

Figure-7. Algorithm to apply type-8 change at node J.

Example-2. Consider the graph of Figure-8. Let all nodes be in Θ_{free} .

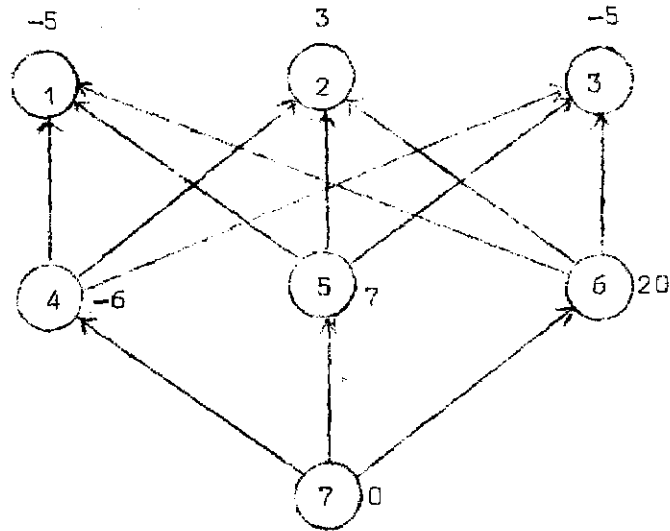


Figure-8

Suppose we apply the heuristic of Figure-5 to DRA_{Θ} corresponding to this graph. The initial solution is as shown in Table-2. The value of $\sum V_j$ for the initial solution is 33.

j	U_j	V_j
1	5	0
2	0	0
3	5	0
4	16	0
5	17	0
6	0	0
7	-	33

Table-2. Initial solution to DRA_{Θ} of the graph in Figure-8.

The first pass through the graph applying Type-A change at each node will result in the improved solution, with $\sum_{j=1}^N V_j = 23$, as shown in Table-3.

The next pass, in which a type-B change is attempted at each node, does not bring about any change in the solution. Hence we stop with the solution shown in Table-3.

J	U_j	V_j
1	0	5
2	0	0
3	0	5
4	0	6
5	0	7
6	0	0
7	-	0

Table-3. Final solution for DRA_{θ} of the graph in Figure-8.

VII. A BRANCH AND BOUND TECHNIQUE FOR MRA_{θ} PROBLEM.

We propose here a branch and bound technique which uses a binary tree [2]. The branching variables are Y_j 's. At each node in the branch and bound tree, we have some Y_j 's fixed to 1, some to zero and the others free. The two descendants for a node are obtained by fixing one of the free Y_j 's to 1 in one descendent, and to 0 in the other. The very first candidate problem in the branch and bound method corresponds to MRA problem, whereas the other candidate problems are of the type MRA_{θ} , (2) - (6).

Bounding. When we pick a candidate problem, we first develop a feasible solution to it, using the heuristic of section V, because the candidate problem is of the form MRA_{θ} . The solution so generated is feasible to MRA also, because MRA_{θ} is a restricted version of MRA . Thus at each node, a feasible solution to MRA is generated, and the best among the feasible solutions generated becomes the incumbent. The incumbent gets updated as the procedure progresses.

For each candidate problem MRA_{θ} , the corresponding DRA_{θ} is solved using the algorithm of Figure-5. Let the value of the solution to DRA_{θ} thus generated be $V(DRA_{\theta})$. $V(DRA_{\theta})$ is a lower bound on the optimal value of MRA_{θ} ; hence, if $V(DRA_{\theta})$ is greater than the value of the incumbent the candidate under consideration can be fathomed; otherwise, we need to branch from the candidate.

Branching. In choosing a Y_j for branching from the candidate problem, we make use of the following complementary slackness conditions which correspond to the primal-dual pair of problems PRA and DRA .

$$U_j - Y_j + \sum_{i \in P(j)} Y_i = 0 \quad j \notin F \quad (19)$$

$$V_j (1 - Y_j) = 0 \quad \text{all } j \quad (20)$$

$$Y_j (a_j + U_j + V_j) = 0 \quad j \in L \quad (21)$$

$$Y_j \left(a_j + U_j + V_j - \sum_{i \in P(j)} U_i \right) = 0 \quad j \notin F, j \notin L \quad (22)$$

$$Y_j \left(a_j + V_j - \sum_{i \in P(j)} U_i \right) = 0 \quad j \in F \quad (23)$$

In the current candidate we examine each $J \in \Theta_{\text{free}}$ in increasing order of J . Suppose we find that $K \in \Theta_{\text{free}}$ violates one of the complementary slackness conditions. If K violates the condition (20), then Y_K is forced to 1 in the right branch and to zero in the left; otherwise, if K violates (21), (22) or (23) then Y_K is forced to 0 in the right branch and to 1 in the left; otherwise, if K violates (19) we scan all predecessors of K and choose a node m , such that $Y_m = 1$ in the feasible solution to current candidate and $c_J \geq c_m$; and make $Y(m) = 0$ in the right branch and $Y(m) = 1$ in the left. If all nodes satisfy the complementary slackness conditions, then for branching we select that vertex in Θ_{free} which has the maximum number of successors. If the chosen vertex J has $\alpha(J) > 0$, then Y_J is forced to 0 in the right branch and to 1 in the left; otherwise Y_J is forced to 1 in the right and to zero in the left. The candidate problem for the right branch is entered in the candidate queue first and then the left one.

The order of retrieval of candidates from the branch and bound tree is last-in-first-out. A formal statement of the algorithm is given in Figure-9.

Algorithm 6.1 (Branch and Bound Method for MRA Problem).BEGIN

- . The problem at the very first node is MRA_n where

$$\theta_{in} = \{N\}; \theta_{out} = \emptyset; \theta_{free} = \{1, \dots, N-1\}$$

- . Enter the first node in the candidate queue.
- . Value of incumbent $\leftarrow \infty$.

REPEAT

- . Remove the last candidate (i.e., the candidate that entered the queue most recently) from the candidate queue, and make it the current node.
- . Attempt to construct a feasible solution to MRA_{θ} , where θ_{in} , θ_{out} and θ_{free} correspond to the current node. (Algorithm of Section V).

IF there is a feasible solution to MRA_{θ} THEN

IF $V(MRA_{\theta}) <$ value of the incumbent solution THEN

- . Value of incumbent solution $\leftarrow V(MRA_{\theta})$
- . Incumbent solution \leftarrow solution obtained for MRA_{θ}

ENDIF

- . Construct a feasible solution to DRA_{θ} (Algorithm of Figure-5)

IF $V(DRA_{\theta}) >$ value of the incumbent solution THEN

- . Fathom the current node

(contd.)

Figure-9

```

ELSE
  IF  $\Phi_{\text{root}}$  is not empty THEN
    . Branch from the current node (Algorithm 5.7)
  ELSE
    . Fathom the current node
  ENDIF
ENDIF
ELSE
  (There is no feasible solution to  $\text{MRA}_{\theta}$  )
  Fathom the current node
ENDIF
UNTIL the candidate queue is empty
IF value of incumbent =  $\infty$  THEN
  There is no feasible solution to the MRA problem.
ELSE
  Optimal solution  $\leftarrow$  incumbent solution
ENDIF
END

```

Figure-9. Branch and Bound Algorithm for MRA problem.

VIII. COMPUTATIONAL EXPERIENCE

We report here briefly the performance of the algorithm in finding MRA for acyclic graphs ranging from four vertices to thirty vertices (Table-4). The algorithm was coded in Fortran-V and executed on CYBER-74 system at Georgia Tech. Within the range of the testing, most problems could be solved in a fraction of a second; however, as this is the first ever attempt to solve this problem, we cannot judge whether the performance reported here is efficient. Though our computational experience is too limited to draw any specific conclusions, the following two observations are worth mentioning: (1) the execution time of the algorithm, as expected, increases with the size of the problem, and (2) when used for solving uncapacitated facility location problems, our algorithm proved inferior to Erlenkotter's (Erlenkotter, 1978). In Table-4, serial number 10 corresponds to a facility location problem with 4 sites and 4 markets. A four-site, five-market problem could not be completed in thirty seconds. This is because, as we increase the number of sites or markets in UL, the number of nodes and arcs in $G(UL)$ increases greatly.

Serial Number	Size of the graph		Number of Nodes in Branch and Bound Tree	Execution Time in seconds.
	Number of vertices	Number of Arcs.		
1	4	5	1	0.01
2	6	9	1	0.018
3	7	10	5	0.09
4	9	21	7	0.196
5	10	18	1	0.04
6	10	13	1	0.04
7	18	28	1	0.05
8	19	21	9	0.38
9	21	30	7	0.28
10	24	40	27	1.5
11	30	30	3	0.7
12	30	30	3	0.7
13	30	30	17	3.5
14	30	30	5	1.0
15	30	36	1	0.442
16	30	36	1	0.442

Table-4. Computational Results of the Branch and Bound Algorithm.

Concluding Remarks

We studied in this paper the MRA problem defined on an acyclic graph with weights on nodes. It is also possible to define a corresponding problem for a rooted graph with weights on arcs. Such a problem can be solved by developing a suitable transformation of the weights-on-arc graph to weights-on-nodes graph, or by modifying the algorithms of this paper to deal with the weights-on-arcs case directly.

REFERENCES

1. Erlenkotter, D. (1978), "A Dual-Based Procedure for Uncapacitated Facility Location", Operations Research, Vol. 26, No. 6, November-December 1978, pp. 992-1009.
2. Geoffrion, A.M., R.E. Marsten (1972), "Integer Programming Algorithms : A Framework and State-of-the-Art Survey", Management Science, Vol. 18, No. 9, May 1972, pp. 465-491.
3. Minieka, E. (1978), "Optimization Algorithms for Networks and Graphs", Marcel Dekker, Inc., New York, 1978.
4. Venkata Rao, V., "Optimal Lot Sizing for Acyclic Multi-Stage Production Systems", Ph.D. Dissertation, Georgia Institute of Technology, University Microfilms Order No. 8124 295 (1981).