ORSA Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

An Efficient Algorithm for the Min-Sum Arborescence Problem on Complete Digraphs

Matteo Fischetti, Paolo Toth,

To cite this article:
Matteo Fischetti, Paolo Toth, (1993) An Efficient Algorithm for the Min-Sum Arborescence Problem on Complete Digraphs. ORSA Journal on Computing 5(4):426-434. http://dx.doi.org/10.1287/ijoc.5.4.426

ORSA Journal on Computing
Vol 5, No 4, Fall 1993

0899-1499/93/0504-0426 $ $01 25

An Efficient Algorithm for the Min-Sum Arborescence Problem on Complete Digraphs

MATTEO FISCHETTI / *DEI, University of Padova, Via Gradenigo G/A, 35100 Padova, Italy*

PAOLO TOTH / *DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy, Email paolo@boder1 cineca it*

(Received July 1991, final revision received September 1992, March 1993, accepted March 1993)

An efficient algorithm for the solution of the Min-Sum Arborescence Problem with fixed root-vertex in complete digraphs is presented The algorithm is based on the well-known Edmonds method The new approach makes use of simple data structures leading to improvements affecting both computing times and memory requirements. Further improvements are obtained by using a new algorithm based on the solution of a sparse problem The linear programming reduced costs associated with the arcs are also computed. A FORTRAN implementation is described; the corresponding code is available, on request, from the authors Extensive computational results on both real-world and random instances are given, showing the effectiveness of the proposed algorithms

The Problem

Let $G = (V, A)$ be a given complete digraph, where $V = \{1, \ldots, n\}$ is the vertex set, and $A = \{(i, j)\ i, j \in V\}$ is the arc set For each arc (i, j) let $c_{i,j}$ be the associated *cost* Given a distinguished *root* vertex r, a *Spanning Arborescence rooted at r* (SA_r) is a partial digraph $\bar{G} = (V, \bar{A})$ such that

- $|\bar{A}| = n - 1$,
- for each vertex $h \in V' = V \setminus \{r\}$, a path from r to h exists in $\bar{G}$ (i e , digraph $\bar{G}$ is *r-connected*)

The *Shortest Spanning Arborescence rooted at r Problem* (SSA_r) consists in finding an $SA_r, G^* = (V, A^*)$, of G whose *cost* $\sum_{(i,j)\in A^*} c_{i,j}$ is a minimum

SSA_r finds many practical applications, e g in network design In addition, it arises as a subproblem in several routing and scheduling problems in which the cost matrix c is often very dense and sometimes complete (for instance, this is the case when the costs satisfy the triangle inequality) As an example, Fischetti and Toth[6] used SSA_r as a relaxation of the well-known Asymmetric Traveling Salesman Problem (ATSP), and report computational results on real-world ATSP instances defined on complete digraphs

A polynomial algorithm for SSA_r was proposed by Edmonds[4] and, independently, by Chu and Liu[3] and by Bock [1] Tarjan[11] gave efficient implementations of the Edmonds algorithm, requiring $O(n^2)$ time for complete digraphs, and $O(|A|\log n)$ time for sparse digraphs Camerini, Fratta and Maffioli[2] repaired an error in Tarjan's implementation Different implementations for sparse digraphs, based on sophisticated data structures, requiring $O(n \log n + |A| \log\log\log_{(|A|/n+2)} n)$ and $O(n \log n + |A|)$ time, have been proposed by Gabow, Galil and Spencer,[9] and by Gabow, Galil, Spencer and Tarjan,[10] respectively

Some definitions and a linear programming mathematical model are given in Section 1, while Edmonds' algorithm is outlined in Section 2 A new $O(n^2)$ implementation of Edmonds algorithm for complete digraphs is described in Section 3 In Section 4 we address the problem of determining the linear programming reduced costs associated with the optimal solution of SSA_r, and propose an efficient $O(n^2)$ algorithm for their computation These reduced costs are useful for performing sensitivity analysis, and can be exploited to improve on lower bounds in those minimization hard problems for which SSA_r is a relaxation (as, for example, the Asymmetric Traveling Salesman Problem and its generalizations[5,6]) Section 5 describes a FORTRAN ANSI package implementing the algorithms of Sections 3 and 4 In Section 6 extensive computational results on large-size real-world and randomly-generated test problems are given, comparing the performances of FORTRAN implementations of the Tarjan[11] algorithm, of the algorithm presented in Section 3, and of a modified version of it involving the transformation of the input complete cost matrix into a sparse one A preliminary version of this paper was presented at the EURO VIII Congress, Lisbon, September 1986

1. Mathematical Models

Given a nonempty vertex set $S \subseteq V \setminus \{r\}$, let $K = (V \setminus S, S)$ denote the *directed r-cutset* (or *r-cut*) containing the arcs $(i, j) \in A$ having $i \in V \setminus S, j \in S$ We denote by $\mathcal{K}$ the family of all r-cuts of digraph G

Given a partial digraph $G_0 = (V, A_0)$ of G, a *strong component* of G_0 is a maximal (with respect to set inclusion) vertex set $S \subseteq V$ such that (i) $|S| = 1$ or (ii) for each pair of distinct vertices i and j in S, at least one path exists in G_0 from vertex i to vertex j

Subject classifications Analysis of algorithms, Networks/Graphs, Transportation

SSA_r can be formulated as the integer linear program

$$v(SSA_r) = \min \sum_{(i,j)\in A} c_{i,j}x_{i,j} \quad (1)$$

$$\text{subject to} \quad \sum_{(i,j)\in K} x_{i,j} \geq 1, \quad \text{for each } K \in \mathscr{K}, \quad (2)$$

$$\sum_{(i,j)\in A} x_{i,j} = n - 1, \quad (3)$$

$$x_{i,j} \in \{0,1\}, \quad \text{for each } (i,j) \in A, \quad (4)$$

where $x_{i,j} = 1$ if arc (i,j) is in the optimal solution G^*, $x_{i,j} = 0$ otherwise

Constraints (2) ensure that G^* is r-connected Without loss of generality, throughout this paper we assume $c_{i,i} = \infty$ for each $i \in V$ and, because of constraint (3), $c_{i,j} > 0$ for each $(i,j) \in A$ In this hypothesis, constraint (3) becomes redundant, and constraints (4) can be relaxed to

$$x_{i,j} \geq 0, \quad \text{for each } (i,j) \in A, \quad (5)$$

thus producing the linear programming formulation (1), (2) and (5) for SSA_r [4 8]

The corresponding linear programming dual problem (D_r) is

$$v(D_r) = \max \sum_{K\in\mathscr{K}} u_K \quad (6)$$

$$\text{subject to} \quad c_{i,j} - \sum_{\substack{K\in\mathscr{K}\\(i,j)\in K}} u_K \geq 0, \quad \text{for each } (i,j) \in A, \quad (7)$$

$$u_K \geq 0, \quad \text{for each } K \in \mathscr{K} \quad (8)$$

Necessary and sufficient conditions for the optimality of $(x^*_{i,j})$ and (u^*_K) for problems SSA_r and D_r, respectively, are then

(i) primal solution $(x^*_{i,j})$ satisfies constraints (2) and (5),
(ii) $u^*_K \geq 0$ for each $K \in \mathscr{K}$, and *reduced cost* $c^*_{i,j} = c_{i,j} - \sum_{K\in\mathscr{K}, (i,j)\in K} u^*_K$ is non-negative for each $(i,j) \in A$,
(iii) $c^*_{i,j} = 0$ for each (i,j) such that $x^*_{i,j} > 0$,
(iv) $\sum_{(i,j)\in K} x^*_{i,j} = 1$ for each $K \in \mathscr{K}$ such that $u^*_K > 0$

2. Edmonds' Algorithm

A pair of solutions $(x^*_{i,j})$ and (u^*_K), satisfying conditions (i) to (iv) above, can be obtained through the following algorithm (for further details, see [4, 8])

The approach is dual feasible, in the sense that, at each iteration, dual feasibility conditions (ii) are satisfied, while primal feasibility is attained only at the last iteration The algorithm is subdivided into two phases

In phase 1, a pair $(x^*_{i,j})$ and (u^*_K) is built satisfying conditions (i) to (iii) Initially all primal and dual variables $x^*_{i,j}$ and u^*_K are assumed to be zero, so the reduced costs $c^*_{i,j}$ are equal to the original costs $c_{i,j}$ Conditions (ii) and (iii), which are clearly satisfied after the initialization, will be iteratively retained At each iteration h, let A_0 be the set containing the arcs $(i,j) \in A$ such that $x^*_{i,j} = 1$ (and hence, from (iii), $c^*_{i,j} = 0$ for each $(i,j) \in A_0$) If digraph $G_0 = (V, A_0)$ is r-connected, condition (i) is satisfied and the second phase is considered Otherwise, at least one strong component S_h of G_0 exists, such that $r \notin S_h$ and $A_0 \cap K_h = \emptyset$, where $K_h = (V \setminus S_h, S_h)$ Then an arc $(i_h, j_h) \in K_h$ having minimum current reduced cost is determined, $x^*_{i_h j_h} = 1$ and $u^*_{K_h} = c^*_{i_h j_h}$ are set, reduced costs $c^*_{i,j}$ with $(i,j) \in K_h$ are updated by subtracting value $u^*_{K_h}$, and a new iteration is performed It can easily be verified that after at most $2n - 3$ iterations condition (i) holds

In phase 2, some arcs (i,j) are removed from A_0 by setting $x^*_{i,j} = 0$ so as to satisfy condition (iv) without affecting the r-connectivity of G_0 This is accomplished by considering the selected arcs (i_t, j_t) in reverse order, and by removing from A_0 all the arcs (i_q, j_q) with $q < t$ and such that the corresponding cut K_q contains (i_t, j_t)

Note that, by construction, primal variables $x^*_{i,j}$ are binary, in addition, we have it that $\sum_{(i,j)\in A} x^*_{i,j} = |A_0| = n - 1$

A Pascal-like description of the algorithm is given

Edmonds' algorithm
comment. all primal and dual variables $x^*_{i,j}$ and u^*_K are initially assumed to be zero,
for each $(i,j) \in A$ **do** $c^*_{i,j} = c_{i,j}$,
$A_0 = \emptyset$, $h = 0$,
comment· Phase 1,
while $G_0 = (V, A_0)$ is not r-connected **do**
  begin
    $h = h + 1$,
    find any strong component S_h of G_0 such that $r \notin S_h$ and $A_0 \cap K_h = \emptyset$, where $K_h = (V \setminus S_h, S_h)$,
    determine $(i_h, j_h) \in K_h$ such that $c^*_{i_h j_h} \leq c^*_{i,j}$ for all $(i,j) \in K_h$,
    $u^*_{K_h} = c^*_{i_h j_h}$, $x^*_{i_h j_h} = 1$, $A_0 = A_0 \cup \{(i_h, j_h)\}$,
    for each $(i,j) \in K_h$ **do** $c^*_{i,j} = c^*_{i,j} - u^*_{K_h}$
  end,
comment Phase 2,
for $t = h$ **to** 1 **step** -1 **do**
  if $x^*_{i_t j_t} = 1$ **then**
    for each $q < t$ such that $j_t \in S_q$ **do**
      begin
        $x^*_{i_q j_q} = 0$, $A_0 = A_0 \setminus \{(i_q, j_q)\}$
      end

Tarjan[11] gave an $O(n^2)$ time implementation of the Edmonds algorithm for complete digraphs As output the procedure returns the primal optimal solution, but not the dual variables (nor reduced costs) The Tarjan algorithm is stated in a manner that achieves the desired asymptotic efficiency, without being concerned with rules that would improve it in practice The implementation is based on efficient data structures to detect the strong components of the current digraph G_0, and keep track of the arcs $(i,j) \in A$ entering each strong component S (for each external vertex $i \in V \setminus S$, the minimum reduced cost arc (i,j) with $j \in S$ is stored) As for phase 2, Camerini, Fratta and Maffioli[2] proposed an efficient data structure to find all the strong components S_q containing a given vertex j_t, allowing the execution of phase 2 in $O(n)$ time

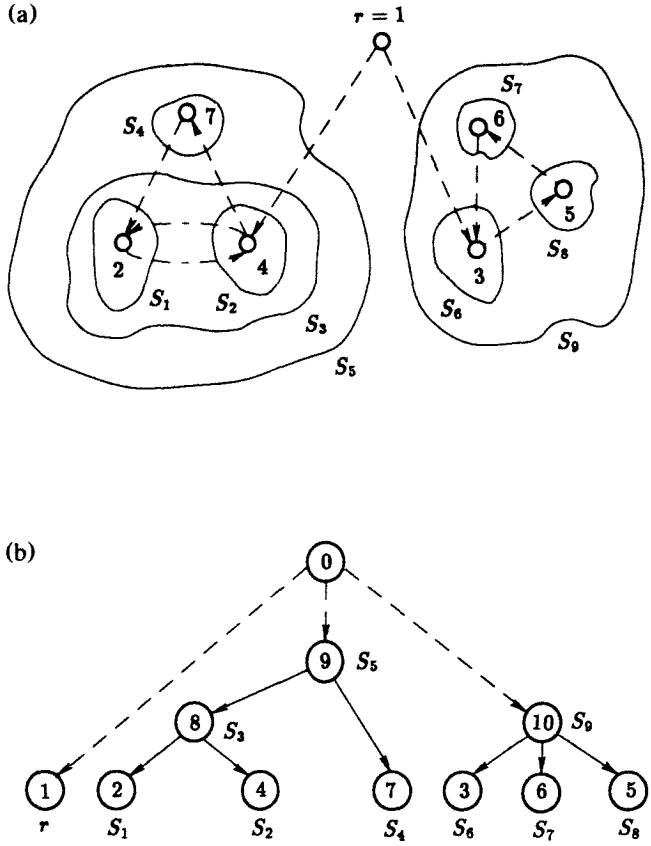

Figure 1. (a) r-cuts generated during phase 1 (dashed line, the arcs of G_0) (b) Auxiliary tree

3. A New Implementation

We now give a different $O(n^2)$ implementation, for complete digraphs, of the Edmonds' algorithm

As for phase 2, we use the data structure proposed by Camerini, Fratta and Maffioli,[2] in which the strong components obtained during phase 1 are represented through the nodes of an auxiliary tree The strong component associated with a node is the union of the strong components associated with the sons of that node in the tree Nodes 1 to n are leaves, and correspond to the original vertices The root node of the auxiliary tree is a dummy node, 0, whose sons are the strong components of digraph G_0 at the end of phase 1 Figure 1 gives a simple instance and the corresponding auxiliary tree We have $n = 7$, $r = 1$, $S_1 = \{2\}$, $S_2 = \{4\}$, $S_3 = \{2,4\}$, $S_4 = \{7\}$, $S_5 = \{2,4,7\}$, $S_6 = \{3\}$, $S_7 = \{6\}$, $S_8 = \{5\}$ and $S_9 = \{3,5,6\}$

As for phase 1, our implementation is based on the following considerations

At any iteration h of phase 1, each strong component S of current digraph G_0 is "shrinkable" into a single "super-vertex" s without affecting the overall computation (multiple arcs incident at s being replaced by their minimum cost arc) Indeed, no matter how cut K_h is chosen, all the vertices of S will belong to the same "side" of the cut Since arc set A_0 increases at each iteration, the above shrinking can be performed in a permanent way In our implementation, as soon as a new strong component S is detected, we replace the rows (columns) of the current reduced cost matrix $(c^*_{i,j})$ associated with the vertices in S with a single row (column) associated with super-vertex s In this way, the size of the current reduced cost matrix decreases during phase 1, allowing a significant reduction in the overall computational effort A second advantage of shrinking is that any new strong component of G_0 corresponds to a circuit, so it can easily be detected

The aim of phase 1 is to have digraph G_0 r-connected To this end we subdivide phase 1 into *stages*, the kth of which introduces in G_0 a path from root vertex r to a given vertex v_k (A similar idea has been proposed, independently, by Gabow, Galil, Spencer and Tarjan[10]) Each vertex v of the current digraph is *labeled* as soon as an arc entering v is introduced in A_0 Initially only vertex r is labeled Let us consider a new stage k, and let v_k be any unlabeled vertex The path from r to v_k is built up in a backward way, and its vertices are stored in a stack (initialized to contain only vertex v_k) Let v be the last vertex inserted in the stack We choose $\{v\}$ as strong component S_h, and find the minimum reduced cost arc (i,v) entering vertex v, which now becomes labeled Three cases can occur

(i) Vertex i has been labeled in a previous stage in this case a path from vertex r to all the vertices in the stack, passing through vertex i, exists in G_0 and the current stage ends
(ii) Vertex i is unlabeled in this case vertex i is inserted in the stack
(iii) Vertex i has been labeled in the current stage in this case the vertices stored in the stack from i to v belong to a circuit of G_0, and hence a new strong component S is detected, S is then shrunk into an unlabeled super-vertex s which replaces the vertices of S in the stack

In order to improve the average behavior of the algorithm, ties in the choice of the minimum reduced cost arc (i,v) are broken by applying in sequence the following rules

(a) choose, if possible, i as a vertex labeled in a previous stage,
(b) choose, if possible, i as an unlabeled vertex,
(c) choose i so as to obtain the largest circuit

These rules prevent equivalent minimum-cost arcs from being selected in the next iterations after one (or more) useless, time consuming shrinking

A Pascal-like description of the algorithm follows

algorithm ARBOR
input: n = number of vertices,
c = cost matrix (with $c_{i,i} = \infty$ for each vertex i),
r = root vertex,
output z^* = cost of the optimal arborescence,
p_j = predecessor of vertex j in the optimal arborescence ($j = 1, \ldots, n$, $p_r = 0$),
main variables
m = number of nodes of the current auxiliary tree (nodes 1 to n correspond to the original vertices, nodes

```
        n + 1 to m to the shrunk strong components, and
        node 0 to the root node),
    LABEL_j = stage in which (super-) vertex j is labeled,
    PARENT_j = parent node of node j in the auxiliary tree,
    U_j = dual variable associated with cut (V \ S, S), where
        S is the strong component corresponding to node j
        of the auxiliary tree (with U_r = 0),
    ARC_j = arc of G_0 entering the strong component corre-
        sponding to node j of the auxiliary tree,
begin
 1  comment: initialization,
 2  m = n, z* = 0,
 3  for j = 1 to n do LABEL_j = PARENT_j = 0,
 4  stage = 1, LABEL_r = stage, U_r = 0,
 5  comment: PHASE 1,
 6  while a vertex h ≤ n having LABEL_h = 0 exists do
      begin
 7      comment a new stage begins,
 8      stage = stage + 1,
 9      initialize the STACK with vertex h,
10      repeat
11        let v be the last vertex inserted in the STACK,
12        find the minimum cost arc (t, v) entering v in
          the current digraph, and let (i, j) be the corre-
          sponding arc in the original digraph (in case of
          ties, rules (a)–(c) are used),
13        LABEL_v = stage,
14        ARC_v = (i, j), U_v = c_{t v}, z* = z* + c_{t v},
15        if LABEL_t = 0 then insert t in the STACK
16        else if LABEL_t = stage then
            begin
17            comment a circuit has been detected,
18            let h_1, ..., h_q be the vertices stored in the
              STACK from h_1 = t to h_q = v,
19            m = m + 1, LABEL_m = PARENT_m = 0,
              c_{m m} = ∞,
20            comment shrink vertices h_1 to h_q into new
              super-vertex m,
21            for each vertex k different from h_1, ..., h_q
              of the current digraph do
                begin
22                let (k, w) be the minimum cost arc of
                  the current digraph with w ∈
                  {h_1, ..., h_q} and define c_{k m} = c_{k w} −
                  U_w,
23                let (w, k) be the minimum cost arc of
                  the current digraph with w ∈
                  {h_1, ..., h_q} and define c_{m k} = c_{w k}
                end,
24            remove h_1, ..., h_q from the STACK, define
              PARENT_{h_1} = ... = PARENT_{h_q} = m, and
              insert m in the STACK
            end,
25      until (LABEL_t ≠ 0) and (LABEL_t ≠ stage)
      end,
26  comment: PHASE 2,
27  p_r = 0,
28  for j = 1 to m do REMOVED_j = false,
29  REMOVED_r = true,
30  for h = m to 1 step − 1 do
31    if not REMOVED_h then
        begin
32        (i, j) = ARC_h,
33        p_j = i,
34        while j ≠ h do REMOVED_j = true, j =
            PARENT_j
        end
end
```

Note that, at step 14, we do not reduce the cost of the arcs entering vertex v (by subtracting value U_v), but postpone this reduction, if necessary, during the shrinking performed at step 22. Also note that it is possible, and convenient, to find the minimum cost arc entering a super-vertex during the corresponding shrinking.

The time complexity of the algorithm above is $O(n^2)$. Indeed, the heaviest computation is performed at steps 12 and 21–23 of phase 1. Step 12 requires $O(n)$ time and is executed $m - 1$ times, with $m \leq 2n - 3$. Each execution of steps 21–23 requires q $O(n)$ time and removes $q - 1$ vertices from the current digraph, so these steps globally require $O(n^2)$ time.

At step 12 the arc (i, j) of the original digraph corresponding to arc (t, v) in the current digraph, is required. So, during shrinking steps 22 and 23 it is necessary to store the arcs of the original digraph corresponding to the selected minimum cost arcs. In order to limit the core memory requirement, we use the following data structure. We assume that, as in all practical cases, any vertex index—i.e., any positive integer not greater than n—can be stored in a single entry of cost matrix c. (If c is real this means that n, represented as a real value, can be stored in a single entry of c.) For each (super-) vertex h of the current digraph we store in row and column $LINE_h$ of matrix c the current cost of the arcs of the current digraph incident at h (with $LINE_h = h$ if h is an original vertex). Let $k = LINE_h$ be the *line* (row/column) associated with a given (super-) vertex h, we define $SHADOW_k = 0$ if h is an original vertex, and $SHADOW_k =$ (pointer to a line of c not used to store arc-costs of the current digraph) if h is a super-vertex. Consider any arc $(h1, h2)$ of the current digraph and let $K1 = LINE_{h1}$, $K2 = LINE_{h2}$, $S1 = SHADOW_{K1}$ and $S2 = SHADOW_{K2}$. the current cost of arc $(h1, h2)$ is stored in $c_{K1\ K2}$, while $i = c_{S1\ K2}$ (with $i = h1$ if $S1 = 0$, i.e., if $h1$ is an original vertex) and $j = c_{K1\ S2}$ (with $j = h2$ if $S2 = 0$) define the arc (i, j) of the original digraph corresponding to arc $(h1, h2)$. During the shrinking of a strong component, the lines of matrix c associated with the shrunk vertices become available. Two of such lines, say k and s, are associated with the new super-vertex m by setting $LINE_m = k$ and $SHADOW_k = s$, and the corresponding entries of c are accordingly defined.

In this way, no extra memory is required to store and update the current digraph, and the overall space complexity of algorithm ARBOR is $n^2 + O(n)$.

4. An Efficient Computation of the Reduced Costs

We now present an $O(n^2)$ algorithm to compute the linear-programming reduced costs associated with the optimal arborescence (see Section 1)

Given the optimal dual variables u_K^* associated with each r-cut $K \in \mathcal{K}$, the reduced cost of arc (i, j) is defined as $c_{ij}^* = c_{ij} - \sum_{K \in \mathcal{K}\,:\,(i,j) \in K} u_K^*$. For instance, in Figure 1a, the cost of arc (3, 2) is decreased by the dual variables associated with r-cuts $(V \setminus S_1, S_1)$, $(V \setminus S_3, S_3)$ and $(V \setminus S_5, S_5)$, while the cost of arc (7, 2) by those associated with r-cuts $(V \setminus S_1, S_1)$ and $(V \setminus S_3, S_3)$. In algorithm ARBOR, each r-cut $K = (V \setminus S, S)$ which can have a positive associated dual variable is represented by the node v of the auxiliary tree corresponding to strong component S, and dual variable u_K^* is stored in U_v. So arc (i, j) belongs to the r-cuts associated with the nodes of the auxiliary tree following the nearest common ancestor of nodes i and j in the path from root node 0 to node j. For instance, in Figure 1 arc (3, 2) belongs to the r-cuts associated with nodes 9, 8 and 2, while arc (7, 2) to those associated with nodes 8 and 2, therefore, we have $c_{32}^* = c_{32} - (U_9 + U_8 + U_2)$ and $c_{72}^* = c_{72} = c_{72} - (U_8 + U_2)$.

A straightforward $O(n^3)$ algorithm for computing the reduced costs would determine for each arc $(i, j) \in A$ the corresponding $O(n)$ crossed r-cuts (through the auxiliary tree) and subtract from c_{ij} the associated dual variables.

A more efficient $O(n^2)$ algorithm is now outlined.

First a permutation $(N_1, \ldots, N_n)$ of the integers $(1, \ldots, n)$ is determined, where node N_{i+1} follows node N_i by scanning the leaves of the auxiliary tree from the left to the right. For instance, in the auxiliary tree of Figure 1b we have $N = (1, 2, 4, 7, 3, 6, 5)$. It is now possible to define, for each non-leaf node h of the auxiliary tree, two integers L_h and R_h (with $L_h < R_h$) such that a leaf node N_j belongs to the subtree having node h as the root if and only if $L_h \leq j \leq R_h$ holds. For instance, in the auxiliary tree of Figure 1b we have $L_8 = 2$, $R_8 = 3$, $L_9 = 2$, $R_9 = 4$, $L_{10} = 5$, $R_{10} = 7$, $L_0 = 1$, $R_0 = 7$, so the leaves of the subtree having node 9 as the root, are nodes N_2, N_3 and N_4, i.e., 2, 4 and 7.

Vectors N, L and R can easily be obtained in $O(n)$ time by applying a standard depth-first digraph search algorithm, which defines the next entry of vector N each time a leaf is visited, value L_h when node h is visited in preorder, and value R_h each time node h is visited in postorder (see Tarjan[12] for more details on digraph search algorithms).

Then reduced costs are computed according to the following scheme. Each vertex j is considered in turn and the reduced cost of all arcs (i, j) entering vertex j are determined. This is accomplished in an efficient way by first considering the arcs whose cost is reduced by U_j, then those whose cost is reduced by $U_j + U_{PARENT_j}$, and so on until all the ancestors of node j are considered.

A Pascal-like description of the algorithm follows.

Algorithm REDUCE
input: n = number of vertices,
c = original cost matrix,
$PARENT_j$ = parent node of node j in the auxiliary tree (0 being the root node),
N, L and R = vectors associated with the auxiliary tree,
U_j = dual variable associated with node j of the auxiliary tree (with $U_r = 0$ for the root vertex r of the arborescence),
output c^* = reduced cost matrix,
begin
  for $k = 1$ **to** n **do**
    begin
      $j = N_k$,
      comment: define reduced costs c_{ij}^* for all vertices i,
      $c_{jj}^* = c_{jj}$,
      $left = right = k$,
      comment: costs c_{ij}^* have been defined for $i = N_{left}, N_{left+1}, \ldots, N_{right}$,
      $\delta = 0$, $h = j$,
      repeat
        $\delta = \delta + U_h$, $h = PARENT_h$,
        for $w = L_h$ **to** $left - 1$ **do** $i = N_w$, $c_{ij}^* = c_{ij} - \delta$,
        for $w = right + 1$ **to** R_h **do** $i = N_w$, $c_{ij}^* = c_{ij} - \delta$,
        $left = L_h$, $right = R_h$
      until $h = 0$
    end
end

Since the repeat-until loop requires $O(n)$ time for each vertex j, the overall time complexity of the algorithm is $O(n^2)$.

5. Program

Algorithms ARBOR and REDUCE were coded in American National Standard FORTRAN as a main subroutine (ARBRED) calling seven subroutines INIT (to initialize the data structures), INSERT (to insert a new vertex in the STACK), NEWST (to initialize a new stage), MINARC (to find the minimum cost arc entering a vertex), SHRINK (to shrink a strong component), ARCARB (to find the arcs of the optimal arborescence), and REDUCE (to compute the reduced cost matrix).

The whole package is completely self-contained and communication to it is achieved solely through the parameter list of ARBRED.

The package can be invoked with the statement

```
CALL ARBRED (N, C, ROOT, ORD, RCFLAG,
    RC, COST, PRED)
```

The input parameters are:
N = number of vertices (n),
C = cost matrix (c) (no restriction on the sign of the entries of C is assumed),
ROOT = root vertex of the arborescence (r),
ORD = dimension of square matrices C and RC,

RCFLAG = flag for the computation of the reduced costs
RCFLAG = 1 if the computation is required, = 0 otherwise

The input-output parameter is

RC = reduced cost matrix (c^*), if RCFLAG = 1

The output parameters are

COST = cost of the optimal arborescence (z^*),

PRED(J) = predecessor of vertex j in the optimal arborescence (p_j, with PRED(ROOT) = 0)

All the parameters are integer After execution, input parameters N, ROOT, ORD and RCFLAG are unchanged On input, matrices C and RC must be equal (both containing the original arc costs) If RCFLAG = 0, arrays C and RC must be coincident, and the package must be invoked through the statement

CALL ARBRED(N, C, ROOT, ORD, 0, C, COST, PRED),

in this case, matrix C is not preserved after execution If RCFLAG = 1, arrays C and RC must be distinct, and the user must copy matrix C into RC before calling the package, in this case, matrix C is preserved after execution

Arrays C and RC must have dimensions ORD × ORD (with ORD ⩾ N), while array PRED must have at least dimension N The package also uses internal integer arrays globally requiring 18N words The internal arrays are currently dimensioned to allow problems for which N ⩽ 1000

The package has been tested on a Digital VaxStation 2000, on an HP 9000/840, and on an IBM PC

FORTRAN code ARBRED is available, on request, from the authors (through e-mail)

6. Computational Results

In this section we experimentally compare code ARBRED with code TARJAN, our FORTRAN ANSI implementation of the $O(n^2)$ algorithm of Tarjan [11] Code TARJAN uses an additional $n \times n$ matrix, INDEX, to keep track of the arcs entering the strong components Each strong component S_h with $|S_h| \geqslant 2$ of the current digraph is associated with two distinct columns of INDEX one giving, through pointers, the vertices i external to S_h, and the other storing for each such i the vertex $j \in S_h$ corresponding to the minimum-cost arc from i to S_h During phase 1 no particular rule is used to choose the strong component S_h and the minimum-cost entering arc As in ARBRED, the minimum-cost arc entering a super-vertex is found during the corresponding shrinking

In addition, we have implemented in FORTRAN ANSI a modified version, SPARB, of algorithm ARBRED, based on the transformation of the input complete cost matrix into a sparse one Indeed, given an optimal arborescence $G^* = (V, A^*)$ let the arc $(i, j) \in A^*$ correspond to the k_jth smallest entry in the column j of the cost matrix ($j \in V \setminus \{r\}$), and define $k^* = \max\{k_j \mid j \in V \setminus \{r\}\}$. An experimental analysis on several classes of test problems showed that k^* is very small and increases only slightly with n Therefore a large portion of the cost matrix can be ignored without affecting the optimality of G^*, with a considerable saving in the computing time The modified algorithm SPARB is as follows

Step 1 For each column $j \neq r$ of the input cost matrix, a threshold $T_j = min_j + \pi(average_j - min_j)$ is first defined, where min_j and $average_j$ are approximations of values $\min\{c_{i,j} \mid i = 1, \ldots, n, i \neq j\}$ and $\Sigma_{i, i \neq j} c_{i,j}/(n-1)$ obtained by sampling the entries of column j, while π is a given parameter For each column, the sampling is performed by taking the $\lfloor n/k \rfloor$ entries in rows $k, 2k, 3k, \ldots$, where k is heuristically set to 10 if $n \leqslant 100$, to $\lceil n/10 \rceil$ otherwise A sparse cost matrix σ is then obtained by taking, for each column j, entries $c_{i,j} < T_j$

Step 2 The optimal arborescence with respect to the sparse cost matrix σ is determined, to this end we do not use an "ad hoc" algorithm for sparse digraphs, but simply modify code ARBRED by introducing, for each vertex j, a *backward star list* containing the pair $(i, c_{i,j})$ for each row i corresponding to an entry of value $c_{i,j}$ in column j of the current sparse cost matrix

Step 3 The optimality of the current solution is checked by computing the complete reduced cost matrix c^* corresponding to the dual solution found at Step 2, and by verifying that condition $c^*_{i,j} \geqslant 0$ holds for all arcs (i, j), if this is not the case, algorithm ARBRED is applied from scratch

According to our computational experience, Steps 1 and 3 are rather time-consuming In order to avoid the execution of Step 3, we introduce a faster, sufficient optimality condition to be checked during Step 2 Let μ_j be a lower bound on the current reduced cost of the arcs entering (super-) vertex j and not belonging to the current sparse cost matrix σ A sufficient optimality condition is then $\mu_j \geqslant 0$ for all current (super-) vertices j Values μ_j can be computed, with overall time complexity $O(n)$, as follows

(i) during step 1 initialize $\mu_j = T_j$ for $j = 1, 2, \ldots, n$,

(ii) when (super-) vertices $h_1, h_2, \ldots, h_q$ are shrunk into the new super-vertex m, define $\mu_m = \min\{\mu_{h_1}, \mu_{h_2}, \ldots, \mu_{h_q}\}$,

(iii) when arc (t, v) is selected as the minimum cost arc entering (super-) vertex v, define $\mu_v = \mu_v -$ (current reduced cost of arc (t, v)), if $\mu_v < 0$ then the current dual solution is not assured to be feasible, and execution of Step 2 is stopped

Algorithms TARJAN, ARBRED and SPARB have been experimentally compared on the following classes of randomly-generated test problems

Class A $c_{i,j}$ uniformly random in range (1–1000),

Class B $c_{i,j}$ uniformly random in range (1–100),

Class C $c_{i,j} = \gamma_{i,j} + \alpha_{i,j}$ with $\gamma_{i,j} = \gamma_{j,i}$ uniformly random in range (1–1000), and $\alpha_{i,j}$ uniformly random in range (1–20)

Class D $c_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + \alpha_{i,j}$, with (x_i, y_i) uniformly random points inside a 1000×1000 square, and $\alpha_{i,j}$ uniformly random in range (1–20)

Class E $c_{i,j}$ as in Class D, with points (x_i, y_i) grouped into $\lfloor n/10 \rfloor$ clusters Each cluster is a square whose center is a random point inside the 1000×1000 square, all clusters have the same area, $1000^2/n$, and globally cover 1/10 of the big square

All costs have been truncated so as to obtain integer values

Instances of classes A and B are completely asymmetric, while those of classes C, D and E are "almost-symmetric" and are obtained through perturbation of symmetric matrices Classes D and E correspond to real-world cases, in which the points to be connected are distributed in a geographical region

For each class, eight different values of n have been considered (n = 50, 100, 200, 300, 400, 500, 800, 1000), for each value of n and for each class, 10 instances were solved on a Digital VaxStation 2000 computer (with fixed working set) Parameter π, used at step 1 of algorithm SPARB, was heuristically set to $4/\sqrt{n}$ (in this way we expect to have approximately $2\sqrt{n}$ entries in each column) With this choice, the optimality test imbedded at step 2 was always satisfied, with the exception of one instance for class D and n = 200, 300, 400 and 800

Tables I–V give the average running times (expressed in seconds) of the three algorithms for classes A, B, C, D and E, respectively As for algorithm ARBRED, we give the running times both to determine the optimal arborescence and to compute the corresponding reduced cost matrix

The results show that algorithm ARBRED strictly dominates that of Tarjan, mainly for instances of class B Problems of classes D and E are harder than those of the other classes for all the algorithms, since more shrinkings are generally required As for classes A and B, we can note that instances of class B are much easier than those of class A for algorithm ARBRED, while the same does not hold for algorithm TARJAN This behavior can be explained by considering that many equivalent minimum-cost arcs exist for class B problems, and that the tie-break rules introduced in ARBRED (and SPARB) avoid several useless shrinkings, which are instead performed by algorithm TARJAN In order to evaluate the dependence of the algorithm performances on the cost range, we considered test problems generated as in classes A and B, with $c_{i,j}$ uniformly random in range $(1, R)$, where R = 25, 100, 250, 1000, 2500, 10000, 25000, and n = 800 Table VI gives the corresponding average computing times, showing that there is no significant difference on the behavior of each algorithm for $R > 1000$

Algorithm SPARB has practically the same performance as ARBRED for problems of classes A and B, while it is clearly superior (when $n \geq 200$) for problems of classes C, D and E Indeed, a significant part of the overall computing time of SPARB is spent, at Step 1, for the sparse cost matrix set-up Since this time is rather independent of the instance to be solved (e g , Step 1 takes about 2 7 seconds and 11 1 seconds for problems with n = 500 and n = 1000, respectively), the harder the problem, the greater the advantage in using algorithm SPARB

As for the reduced cost matrix computation, the results show that running time is practically independent of problem generation In order to evaluate the effectiveness of the technique for the reduced costs computation, we imple-

Table I. Problems of Class A ($c_{i,j}$ Uniformly Random in (1–1000)) Average Running Times over 10 Instances (in VaxStation-2000 Seconds)

	Optimal Arborescence			Reduced Costs
n	TARJAN	ARBRED	SPARB	ARBRED
50	0 11	0 07	0 12	0 04
100	0 39	0 25	0 32	0 12
200	1 59	1 03	1 01	0 45
300	3 07	2 01	1 85	0 97
400	5 35	3 57	3 06	1 84
500	9 88	6 63	5 27	2 85
800	21 15	14 68	11 91	9 01
1000	41 92	28 59	17 28	13 27

Table II Problems of Class B ($c_{i,j}$ Uniformly Random in (1–100)) Average Running Times over 10 Instances (in VaxStation-2000 seconds)

	Optimal Arborescence			Reduced Costs
n	TARJAN	ARBRED	SPARB	ARBRED
50	0 11	0 06	0 10	0 03
100	0 42	0 22	0 29	0 11
200	1 65	0 79	0 82	0 43
300	2 97	1 25	1 33	0 83
400	5 54	2 03	2 09	1 42
500	9 47	3 00	3 07	2 34
800	27 57	7 99	8 21	7 01
1000	40 97	13 03	12 67	11 55

Table III Problems of Class C ($c_{i,j}$ Almost Symmetric) Average Running Times over 10 Instances (in VaxStation-2000 Seconds)

	Optimal Arborescence			Reduced Costs
n	TARJAN	ARBRED	SPARB	ARBRED
50	0 21	0 15	0 20	0 04
100	0 70	0 51	0 53	0 12
200	2 58	1 82	1 59	0 58
300	5 22	3 70	2 83	1 12
400	9 22	6 53	4 77	2 23
500	12 64	8 82	6 07	2 94
800	30 04	21 57	15 05	9 59
1000	60 63	39 13	23 78	12 39

Table IV. Problems of Class D ($c_{i,j}$ Almost Euclidean) Average Running Times over 10 Instances (in VaxStation-2000 Seconds)

	Optimal Arborescence			Reduced Costs
n	TARJAN	ARBRED	SPARB	ARBRED
50	0 27	0 17	0 19	0 03
100	1 04	0 66	0 58	0 12
200	4 01	2 55	1 84	0 42
300	9 02	5 63	3 59	0 90
400	15 60	9 92	5 88	1 62
500	24 50	15 17	7 24	2 76
800	63 94	39 04	21 67	9 20
1000	96 31	62 14	26 78	12 69

Table V Problems of Class E ($c_{i,j}$ Almost Euclidean with $\lfloor n / 10 \rfloor$ Clusters) Average Running Times over 10 Instances (in VaxStation-2000 Seconds)

	Optimal Arborescence			Reduced Costs
n	TARJAN	ARBRED	SPARB	ARBRED
50	0 28	0 17	0 16	0 03
100	1 02	0 63	0 47	0 12
200	4 03	2 44	1 42	0 41
300	8 85	5 34	2 76	0 88
400	15 51	9 39	4 53	1 50
500	24 37	14 54	6 82	2 53
800	63 01	36 90	17 62	7 90
1000	97 42	58 10	28 03	11 48

Table VI. Problems with $c_{i,j}$ Uniformly Random in Range $(1 - R)$, $n = 800$ Average Running Times over 10 Instances (in VaxStation-2000 Seconds)

	Optimal Arborescence			Reduced Costs
R	TARJAN	ARBRED	SPARB	ARBRED
25	29 52	6 61	8 05	6 78
100	27 57	7 99	8 21	7 01
250	24 58	10 47	9 05	7 55
1000	21 15	14 68	11 91	9 01
2500	21 89	16 04	12 71	9 00
10000	22 11	16 67	13 25	9 11
25000	21 19	15 97	12 73	8 88

mented in FORTRAN the following simple benchmark algorithm to simulate the computations involved in the reduced cost matrix definition

for $j = 1$ **to** n **do for** $k = 1$ **to** n **do** $i = pointer_k$, $c^*_{i,j} = c_{i,j} - value$

where $pointer_k = n - k + 1$ and $value = 1$ have been previously defined In our opinion no reduced cost algorithm

Table VII Real-World Instances, Running Times Are in VaxStation-2000 Seconds

		Optimal Arborescence			Reduced Costs
NAME (n)		TARJAN	ARBRED	SPARB	ARBRED
RY48P	(48)	0 21	0 16	0 18	0 03
RBG050A	(50)	0 14	0 09	0 12	0 03
FT53	(53)	0 20	0 12	0 15	0 04
FT70	(70)	0 31	0 27	0 49	0 09
FTV70	(71)	0 26	0 18	0 23	0 06
KRO124P	(100)	0 78	0 54	0 44	0 11
FTV180	(181)	1 83	1 20	0 78	0 31
RBG323A	(323)	5 50	3 34	2 84	1 90
RBG378A	(378)	4 67	2 88	2 27	1 48
RBG443A	(443)	6 28	3 78	3 01	2 19

faster than this benchmark can be designed The computing time of the benchmark algorithm on the problems of classes A to E is approximately 20% smaller than that of algorithm REDUCE, on average (e g , the benchmark algorithm takes 2 17 and 6 68 seconds for problems with $n = 500$ and $n = 800$, respectively)

In addition, we have analyzed the computational behavior of algorithms TARJAN, ARBRED, and SPARB on 10 real-world ATSP instances Problem RY48P, FT53, FT70, and KRO124P are from Fischetti and Toth [6] Problems FTV70 and FTV180 are taken from Fischetti, Toth, and Vigo[7] and refer to pharmaceutical product delivery in downtown Bologna, the remaining instances are scheduling problems from Siemens Nixdorf, Augsburg (Germany) For all problems, the root of the arborescence is vertex 1 Table VII gives the corresponding computational results, showing that ARBRED and SPARB outperform TARJAN for these real-world instances too

Acknowledgment

Work supported by M U R S T , Italy, and NATO Research Grant No 228/88 Thanks are due to Martin Groetschel and Norbert Ascheuer for having provided us with the real-world ATSP instances from Siemens Nixdorf, Augsburg

References

1 F Bock, 1971 An Algorithm to Construct a Minimum Directed Spanning Tree in a Directed Network, in *Developments in Operations Research*, Gordon and Breach, New York, pp 29–44
2 P M Camerini, L Fratta and F Maffioli, 1979 A Note on Finding Optimum Branchings, *Networks 9*, 309–312
3 Y J Chu and T H Liu, 1965 On the Shortest Arborescence of a Directed Graph, *Sci Sinica 14*, 1396–1400
4 J Edmonds, 1967 Optimum Branchings, *J Res Nat Bur Standards 71B*, 233–240
5 M Fischetti and P Toth, 1988 An Additive Approach for the Optimal Solution of the Prize-Collecting Travelling Salesman Problem, in B L Golden and A A Assad (eds), *Vehicle Routing Methods and Studies*, North Holland, pp 319–343

6 M FISCHETTI and P TOTH, 1992 An Additive Bounding Procedure for the Asymmetric Travelling Salesman Problem, *Mathematical Programming 53*, 173–197
7 M FISCHETTI, P TOTH and D VIGO, 1993 A Branch and Bound Algorithm for the Capacitated Vehicle Routing Problem on Directed Graphs, *Operations Research* (to appear)
8 D R FULKERSON, 1974 Packing Rooted Cuts in a Weighted Directed Graph, *Mathematical Programming 6*, 1–13
9 H N GABOW, Z GALIL and T SPENCER, 1984 Efficient Implementation of Graph Algorithms Using Contraction, in Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, pp 347–357
10 H N GABOW, Z. GALIL, T SPENCER and R.E TARJAN, 1986 Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs, *Combinatorica 6*, 109–122
11 R.E TARJAN, 1977 Finding Optimum Branchings, *Networks 7*, 25–35
12 R.E TARJAN, 1983 Data Structures and Network Algorithms, CBMS 44, Society for Industrial and Applied Mathematics, Philadelphia, PA