Shortest-Path Methods: Complexity, Interrelations and New Propositions

Stefano Pallottino
Consiglio Nazionale delle Ricerche, Istituto per le Applicazioni del Calcolo "Mauro Picone" (IAC), 137, viale del Policlinico, 00161 Roma, Italy

We present a new unified approach for shortest-path problems. Based on this approach, we develop a computational method which consists of determining shortest paths on a finite sequence of partial graphs defined as the "growth of the original graph." We show that the proposed method allows us to interpret within the same framework several different well-known algorithms, such as those of D'Esopo–Pape, Dijkstra, and Dial, and leads to a uniform analysis of their computational complexity. We also stress the existing parallelism between the proposed method and the matrix-multiplication methods of Floyd–Warshall, and Dantzig.

I. INTRODUCTION

Many efficient algorithms for solving shortest-path problems on directed graphs currently exist, as well as an extensive literature that examines their worst-case computational complexities and average performance [6, 13, 17].

Recently Gallo and Pallottino [12] found that all shortest-path tree algorithms are special cases of a more general algorithm; each special case is based on the choice of a different *priority queue* system for handling the labeled nodes of a graph.

This paper aims to provide a new interpretation for these algorithms as well as for algorithms that compute all shortest-path problems. We will refer to a sequence of graphs $G^{(1)}, \ldots, G^{(H)}$, $1 \leqslant H \leqslant n$, such that

$$G^{(h)} \subseteq G^{(h+1)}, \qquad h = 1, 2, \ldots, H - 1.$$

This sequence will be called the "graph growth of G," ($G^{(H)} \equiv G$).

It is shown that solving a shortest-path problem on G is equivalent to solving the same problem on all graphs of the growth, using the solution of the previous graph to solve the next one.

The originality of this paper lies in the interpretation of shortest-path methods as graph-growth methods.

In the following sections, it will be seen that the graph-growth concept provides an easier approach to the interpretation and understanding of the algorithms and also

NETWORKS, Vol. 14 (1984) 257–267

CCC 0028-3045/84/020257-11$04.00

helps in the analysis of their computational complexities. In addition, it becomes possible to compare shortest-path tree methods with matrix-multiplication methods.

Part of this paper is devoted to an analysis of the D'Esopo-Pape algorithm and to the development of a new algorithm of the same type. This new algorithm showed much-improved computational complexity [19].

For the sake of brevity, we will not examine in detail areas which are perfectly well covered by the current literature, but we will merely recall the essential parts. References present material which is not included in this paper and all definitions related to graph theory.

II. THE BELLMAN-FORD-MOORE ALGORITHM

Let $G = (N, A)$ be a directed graph, where $N = \{1, 2, \ldots, n\}$ is the *set of nodes*, and A, $|A| = m$, is the *set of arcs*; with each arc $(i, j) \in A$, $i \in N$, $j \in N$, is associated a finite *length* l_{ij}. Let $r \in N$ be the *root* of the shortest-path (SP) (directed) tree T that we wish to determine. Let d_j be the length of the path in T that connects r to any j. When $j = r$, $d_r = 0$. We denote by S_i and P_i, respectively, the sets of successor and predecessor nodes of i:

$$S_i = \{j : (i, j) \in A\}, \qquad P_i = \{j : (j, i) \in A\}.$$

Without loss of generality, we suppose that G is strongly connected and without negative circuits. The lengths of the SPs are the solution of Bellman's equations [2]:

$$d_r = 0; \qquad d_j = \min\{d_i + l_{ij} : i \in P_j\}, \qquad j \neq r. \tag{1}$$

An iterative solution method for Eqs. (1) was proposed by Ford [10]:

$$\begin{aligned} &d_r^{(0)} = 0; \qquad d_j^{(0)} = +\infty, \qquad j \neq r; \\ &d_j^{(k+1)} = \min\{d_j^{(k)}, \min\{d_i^{(k)} + l_{ij} : i \in P_j\}\}, \qquad j \in N, \end{aligned} \tag{2}$$

where $d_j^{(k)}$ is the length of the SP from r to j which includes at most k arcs. It is obvious that each path of T includes less than n arcs. So we know that $d_j^{(n)} = d_j$, for each $j \in N$. The computational complexity of the algorithm based on method (2) is $O(mn)$. To eliminate unnecessary operations, a *priority queue* (PQ) Q can be introduced (see [1]):

$$\begin{aligned} &Q^{(0)} = \{r\}, \\ &Q^{(k)} = \{i \in N : d_i^{(k)} < d_i^{(k-1)}\}, \qquad k = 1, 2, \ldots, n, \end{aligned} \tag{3}$$

and apply (2) only for the nodes of the PQ:

$$d_j^{(k+1)} = \min\{d_j^{(k)}, \min\{d_i^{(k)} + l_{ij} : i \in P_j \cap Q^{(k)}\}\}, \qquad j \in N. \tag{2'}$$

Ordering the operations differently, (2') corresponds to the following procedure:

$$\begin{aligned} &d_j^{(k+1)} = d_j^{(k)}, \qquad j \in N, \\ &d_j^{(k+1)} = \min\{d_j^{(k+1)}, d_i^{(k)} + l_{ij}\}, \qquad j \in S_i,\ i \in Q^{(k)}. \end{aligned} \tag{2''}$$

An implementation technique for the procedure ($2''$) has been developed by Ford [10] and Moore [18]. In this procedure, Q denotes a *queue* (see [16]), on which the following operations are defined:

QINIT(Q, r)	initialization: $Q = \{r\}$;
QIN(Q, j)	insertion: if $j \notin Q$, j is inserted at the end of the queue;
QOUT(Q, i)	reduction: i is removed from the beginning of the queue;
$Q = \phi$?	check whether $Q = \phi$.

To implement the queue efficiently, a *one-way-linked list* is used because all the above operations have constant computational complexity except QINIT, which has complexity $O(n)$.

In Figure 1, we present a *general SPT* (*Shortest-Path Tree*) algorithm; if the above operations are used, we obtain the Bellman-Ford-Moore algorithm, which we will call SP/queue to emphasize the choice of the queue as PQ.

The computational complexity of SP/queue is $O(mn)$ because each node will be removed from Q no more than n times (see [12]).

Note that complexity is strongly related to the choice of the PQ Q. For example, if we chose Q to be a *stack*, the complexity of the algorithm SP/stack would be $O(n2^n)$.

The procedure SPT assembles and includes most SP tree algorithms; the two major families of methods LC and LS (label correcting and label setting, respectively) can be represented by SPT.

For example, Dijkstra's algorithm [7] is obtained from SPT by taking Q to be a nonordered list; in this case, the QOUT operation consists of finding and removing from Q the node i that is nearest to r (with minimal label):

$$d_i = \min\{d_j : j \in Q\}.$$

All others LS algorithms have the same QOUT operations but they are based on other kinds of PQs (see [4, 6, 12, 13]).

```
Procedure SPT(r);
  begin
    QINIT(Q,r);
    for i := 1 to n do D[i] := +∞;
    D[r] := 0;
    repeat
      QOUT(Q,i);
      foreach j ∈ S[i] do
        if D[j] > D[i] + L[i,j] then
          begin
            D[j] := D[i] + L[i,j];
            QIN(Q,j)
          end
    until Q = ∅
  end;
```

FIG. 1. SPT algorithm.

III. THE D'ESOPO-PAPE ALGORITHM

In 1974 Pape [20] exploited a suggestion of D'Esopo (see [21]) and proposed an algorithm that quickly proved to be the best algorithm for most concrete cases (see [6]). This algorithm corresponds to the general SPT algorithm where Q is a *double-ended queue* (or *deque*) (see [16]). The deque allows the insertion of nodes at both ends of the PQ according to a predetermined strategy. (See Fig. 2.)

A distinction can be made between nodes that are in Q and others: a node i is an element of Q when the current distance d_i from the root has not already been used to find a SP.

D'Esopo's suggestion implemented by Pape was to split the nodes not in Q into two classes: (i) the "unlabeled" nodes, i.e., those that have never entered Q (i.e., whose distance from r is still infinity); (ii) the "labeled and scanned" nodes, i.e., those that have passed through Q at least once, and whose current distance from r has already been used. The unlabeled nodes are inserted at the end of Q, while the nodes which have been labeled and scanned are inserted at the beginning of Q.

An easy approach to the implementation of Q consists of using a code to distinguish between the two classes of nodes and a *one-way-linked list* of pointers with two additional pointers to indicate the two ends of Q:

QIN is the only operation to be changed:

QIN(Q, j)	insertion:	if $j \notin Q$ insert j at the end of Q if it is unlabeled or at the beginning of Q if it has been labeled and scanned.

This type of PQ has a definite impact on the performance of the corresponding algorithm, called SP/deque. When an explored node i is reinserted in Q, this means that its *label* d_i has just been improved.

In the current tree T, there exist a subtree $T(i)$ which has i as a root. The improvement in d_i must be applied to every node in $T(i)$. Node i is inserted at the beginning of Q to hasten its scanning, and, consequently, the update of the labels of the nodes of $T(i)$.

This interest in "updating as quickly as possible" explains the need to distinguish between unlabeled nodes and those which have been labeled and scanned. Note that each node will be inserted at the end of Q only once (i.e., the first time); on subsequent occasions (if these occur), it will be inserted at the beginning of Q. This can be interpreted in two different ways.

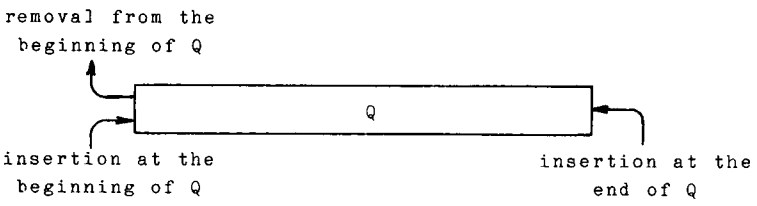

FIG. 2. The deque.

A. The Priority Queue as a Coupling of a Stack and a Queue

Q can be considered as a pair of priority queues Q' and Q'': Q' handles the nodes that are unlabeled when they enter; Q'' handles the labeled nodes. Q' is used for the removal of a node only when Q'' is empty.

It is clear that Q' is used as a queue and Q'' as a stack. In the operation of SP/deque, after the removal of the hth node of Q' ($h = 2, 3, \ldots, n-1$), the algorithm works on $h-1$ nodes with a stack as PQ, i.e., with an SP/stack algorithm, which has a complexity of $O(n2^{h-1})$ in the worst case. Globally, the complexity of SP/deque is therefore $O(n2^n)$. (See Fig. 3.)

B. Graph Growth

A *graph growth of* G is defined to be any sequence of graphs $G^{(1)}, G^{(2)}, \ldots, G^{(H)}$ such that

$$\begin{aligned} &G^{(h)} \subseteq G^{(h+1)}, \quad h = 1, \ldots, H-1, \\ &G^{(H)} \equiv G. \end{aligned} \tag{4}$$

Let $\mathcal{P}(N)$ be a permutation of the elements of N; let $N^{(h)}$ be the set of the first h elements of the permutation $\mathcal{P}(N)$.

If the hth graph of the growth, $G^{(h)}$, is such that

$$A^{(h)} = \{(i,j) : i \in N^{(h)}\}, \quad G^{(h)} = (N, A^{(h)}), \tag{5}$$

the growth is called a *graph growth of* G *by partial graphs*.

Through this graph growth of G, a simpler interpretation of the differentiated treatment of unlabeled and labeled nodes can be given.

Let $\mathcal{P}(N)$ be the permutation corresponding to the order in which nodes are scanned for the first time; let $N^{(h)}$ be the set of all nodes that are removed at least once from Q (nodes which are labeled and scanned and nodes in Q''). Then finding the SP in G is equivalent to finding the SP in the n graphs of the graph growth for the permutation $\mathcal{P}(N)$.

Intuitively, if i_h is the hth node removed from Q', in scanning the *forward star* of i_h, only nodes of $N^{(h)}$ are used in Q''. When $Q'' = \phi$, the resulting tree $T^{(h)}$ is in fact the SP tree of $G^{(h)}$.

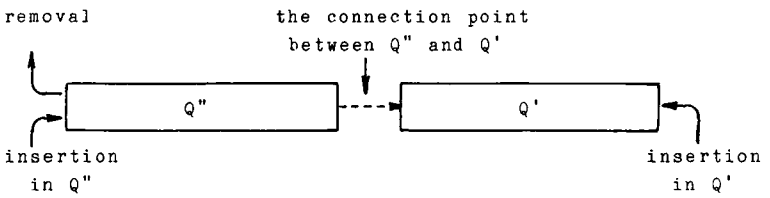

FIG. 3. Deque Q as a pair Q'-Q''.

The hth iteration transforms $T^{(h-1)}$ into $T^{(h)}$ through the graph growth:

$$A^{(h)} = A^{(h-1)} \cup \{(i_h, j) : j \in S_{i_h}\}. \tag{5'}$$

The resulting algorithm can be interpreted as a master algorithm SPT (see Fig. 4) which manages the different iterations on the graph growth and which requires, for each growth $G^{(h)}$, an algorithm SP/base (see Fig. 5) which finds the new tree $T^{(h)}$ from $T^{(h-1)}$.

The computational complexity of algorithm SPT has two components: the complexity of SP/base summed over the different iterations, and the complexity $\gamma(m, n)$ of the initialization and iteration management procedures (QOUT, *until* $Q = \emptyset$).

Let $m^{(h)}$ be the number of arcs of $G^{(h)}$ and $b(m^{(h)}, n)$ be the computational complexity of SP/base at the hth iteration. The computational complexity of SPT is

$$O\left(\sum_{h=1}^{n} b(m^{(h)}, n) + \gamma(m, n)\right). \tag{6}$$

The basic algorithm for SP/deque is SP/stack. At the hth iteration, SP/deque removes node i_h from Q' and calls SP/stack to work with Q'' on the arcs $A(h)$, as in (5'). Depending on whether the nodes are unlabeled or labeled and scanned, SP/stack inserts them into either Q' or Q''.

The computational complexity of SP/stack is $b(m^{(h)}, n) = O(n2^{h-1})$ and so the complexity of SP/deque is $O(n2^n)$ from (6).

C. Other Applications of the Graph-Growth Approach

Any algorithm based on the distinction between labeled and scanned and unlabeled nodes can be transformed into a classic algorithm applied to a graph growth.

Similar interpretations exist for all other algorithms. For example, in the case of Dijkstra's algorithm [7], where the lengths are non-negative, each node will be removed from the PQ only once, (see [15]). Let i_h be the hth node removed from Q. Then the following is true:

$$d_{i_k} \leqslant d_{i_h}, \quad k = 1, \ldots, h-1; \quad d_{i_k} \geqslant d_{i_h}, \quad k = h+1, \ldots, n. \tag{7}$$

```
Procedure SPT(r);
  begin
    QINIT(Q,r);
    for i := 1 to n do D[i] := +∞;
    D[r] := 0;
    repeat
      QOUT(Q,i);
      SPBASE(i)
    until Q = ∅
  end;
```

FIG. 4. Master algorithm SPT.

```
Procedure SPBASE(i);
  begin
    foreach j ∈ S[i] do
      if D[j] > D[i] + L[i,j] then
        begin
          D[j] := D[i] + L[i,j];
          QIN(Q,j)
        end
  end;
```

FIG. 5. SP/base algorithm.

Moreover, we know that any new arc of $T^{(h)}$

$$(i,j) \in T^{(h)} - T^{(h-1)} \tag{8}$$

is an arc of the forward star of i_h, and therefore $(i,j) \in A^{(h)} - A^{(h-1)}$. The graph growth provides a natural and direct interpretation of Dijkstra's algorithm and its properties. The graph growth is given for a permutation corresponding to the topological order of G with respect to r. SP/Dijkstra is a master algorithm whose role is to find at each iteration the node with the minimum label in Q and to call its own SP/base.

SP/base builds $T^{(h)}$ from $T^{(h-1)}$ by exploring the arcs of the hth growth of G. Reoptimizing $T^{(h-1)}$ to find $T^{(h)}$ corresponds to updating $n - h$ labels:

$$d_{i_k} = \min\{d_{i_k}, d_{i_h} + l_{i_h i_k}\}, \quad k = h+1, \ldots, n. \tag{9}$$

Based on properties (7) and (8), the computational complexity of SP/base, for each iteration, is $b(m^{(h)}, n) = O(m^{(h)} - m^{(h-1)})$.

The management section of the master algorithm comprises n searches of minimal values among $n - h$ elements ($h = 1, 2, \ldots, n$), so the complexity of this section is $\gamma(m, n) = O(n^2)$. Globally, from (6), the computational complexity of SP/Dijkstra is $O(n^2)$.

Other algorithms of the LS family can be just as easily interpreted. Dial's algorithm [5], for example, has a different SP/base (which uses an address-calculation array and a two-way-linked list), but its complexity is still $O(m^{(h)} - m^{(h-1)})$. Let $l_{\max}$ be the length of the address array. Then $\gamma(m, n) = O(nl_{\max})$ and the global complexity of SP/Dial is $O(m + nl_{\max})$.

We now consider an algorithm which takes a binary heap as its PQ, (see [24]). The global complexity of SP/heap is $O(m \log(n))$ because the management section is linear in $n \log(n)$ and the corresponding SP/base has complexity $O(\log(n - h)(m^{(h)} - m^{(h-1)})$ for the hth iteration.

Similar interpretations can be developed for other algorithms of the LS family such as those proposed by Denardo and Fox [4] and Van Vliet [22].

Finally, we show that the change of $T^{(h-1)}$ into $T^{(h)}$ caused by the growth of $G^{(h-1)}$ into $G^{(h)}$ can be interpreted in terms of reoptimization. In fact, at the beginning, we merely initialize to infinity the lengths of all arcs; then for the hth growth,

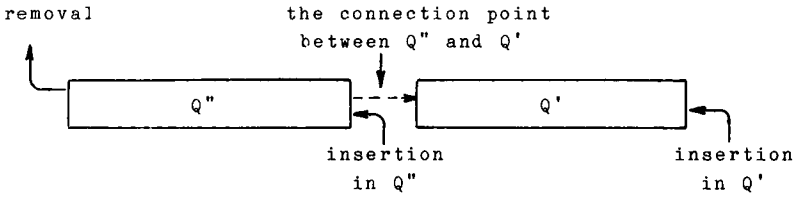

FIG. 6. 2que, or double-queue.

we give back the original lengths to the arcs of the forward star of i_h and use a specific reoptimization algorithm (see [11]).

IV. A NEW ALGORITHM: SP/2QUE

The weak point of SP/deque is that its corresponding SP/base has a stack as a PQ; this gives a worst-case complexity which is exponential with respect to the number of nodes. It is clear that to obtain a more efficient algorithm, the stack must be replaced by another type of PQ. We suggest using a queue instead of a stack; then, the PQ is a sequence of two connected queues. (See Fig. 6.)

The implementation of this algorithm is very simple; one more pointer must be added to the end of the first queue (or the connection point of the two queues). We call this algorithm SP/2que.

The only operation on Q that changes is QIN:

QIN(Q, j) insertion: if $j \in Q$, insert j at the end of Q if it is unlabeled or at the connection point if it has been previously labeled and scanned.

The SP/base of SP/2que is an SP/queue which is applied in the hth iteration on a graph with h nodes and $m^{(h)}$ arcs: $b(m^{(h)}, n) = O(hm^{(h)})$. Iteration management is linear in n, so the computational complexity of SP/2que is

$$O\left(n + \sum_{h=1}^{n} b(m^{(h)}, n)\right) = O\left(n + \sum_{h=1}^{n} hm^{(h)}\right) = O\left(n + m \sum_{h=1}^{n} h\right) = O(mn^2).$$

Thus we obtain much better computational complexity than for SP/deque. Computational tests have demonstrated the real efficiency of SP/2que (see [19]).

V. RELATIONS WITH MATRIX-MULTIPLICATION ALGORITHMS

Let L be the arc length matrix:

$$l_{ij} = \begin{cases} 0, & i = j, \\ l_{ij}, & (i,j) \in A, \\ +\infty, & \text{otherwise.} \end{cases} \tag{10}$$

We introduce the *minsum* operation between two matrices:

$$X = X \oplus Y, \tag{11}$$

such that

$$z_{ij} = \min\{x_{ik} + y_{kj} : k = 1, \ldots, n\}, \quad i, j = 1, \ldots, n. \tag{12}$$

Let $D^{(1)} = L$; the most simplistic matrix-multiplication (MM) method for finding all SPs is the application of n minsum operations to the matrix L:

$$D^{(h)} = D^{(h-1)} \oplus L = L \oplus L \oplus L \oplus \cdots \oplus L \quad (h \text{ times}).$$

This comes directly from the Bellman's equations (1) and from the iterative method of Bellman and Ford:

$$d_{ij}^{(h)} = \min\{d_{ik}^{(h-1)} + l_{kj} : k = 1, \ldots, n\}, \quad i, j = 1, \ldots, n. \tag{12'}$$

In the end we obtain $D^{(n-1)} = D$, the shortest-distances matrix.

This method can be interpreted as n applications of the Bellman-Ford method, with a different node as a root each time; i.e., the Bellman-Ford method builds the shortest-distance matrix, a row at a time.

A more efficient MM method, having a complexity of $O(n^3)$, has been proposed by Floyd [9] (see also [14, 23]). Let $D^{(0)} = L$; the hth iteration is obtained from the previous one by

$$d_{ij}^{(h)} = \min\{d_{ij}^{(h-1)}, d_{ih}^{(h-1)} + d_{hj}^{(h-1)}\}. \tag{13}$$

$d_{ij}^{(h)}$ represents the length of the SP from i to j that does not intersect the nodes $h + 1, \ldots, n$. (This can be easily demonstrated by induction.) Finally, we have $D^{(n)} = D$.

A new interpretation can be given for the Floyd-Warshall method. Consider the rth row of $D^{(h)}$:

$$d_{rj}^{(h)} = \min\{d_{rj}^{(h-1)}, d_{rh}^{(h-1)} + d_{hj}^{(h-1)}\}. \tag{14}$$

This row represents the SP lengths, starting from r, of $G^{(h)}$, the hth growth of G, where $\mathcal{P}(N) = \{r, 1, 2, \ldots, r-1, r+1, \ldots, n\}$.

This results directly from the interpretation given to $d_{ij}^{(h)}$. A SP tree $T^{(h)}$ of $G^{(h)}$ is associated with this row of $D^{(h)}$. It is clear that the Floyd-Warshall algorithm is valid for any $\mathcal{P}(N)$ permutation.

So any algorithm based on a graph growth of G by partial graphs is equivalent to the computation of the rth row of the D matrix according to the Floyd-Warshall method.

Note that at the end of the $(h-1)$th iteration, the Floyd-Warshall algorithm supplies the $(h-1)$th row of $D^{(h-1)}$, which the algorithm based on graph growth cannot do. The task of the SP/base algorithm is in fact the computation of this row—i.e., the calculation of the sum of the row's elements with the current distance from r to i_h. Algo-

rithms of the SP/base type have to recompute intermediate results because of the lack of memory to retain them.

There exists another type of MM method, developed by Dantzig [3], which corresponds to a reorganization of the operations of the Floyd-Warshall method (see also [8]).

Let $d_{ij}^{(h)}$ be the length of the SP from i to j, $i, j = 1, 2, \ldots, h$ that only uses the nodes $1, 2, \ldots, h$; then $D^{(h)}$ is a square matrix of order h. At the beginning, $D^{(1)} : d_{11} = 0$. From $D^{(h-1)}$ we obtain $D^{(h)}$ as follows:

$$\begin{aligned} d_{ih}^{(h)} &= \min\{d_{ij}^{(h-1)} + l_{jh} : j = 1, \ldots, h-1\}, \quad i = 1, \ldots, h-1, \\ d_{hj}^{(h)} &= \min\{d_{ij}^{(h-1)} + l_{hi} : i = 1, \ldots, h-1\}, \quad j = 1, \ldots, h-1, \\ d_{ij}^{(h)} &= \min\{d_{ij}^{(h-1)}, d_{ih}^{(h)} + d_{hj}^{(h)}\}, \quad i, j = 1, \ldots, h-1, \\ d_{hh}^{(h)} &= 0. \end{aligned} \tag{15}$$

It is quite simple to show that the matrix $D^{(n)} = D$.

Consider now a new graph growth: *the growth of* G *by subgraphs:* $G^{(1)}, G^{(2)}, \ldots, G^{(n)}$ such that the subgraph $G^{(h)}$ is described by

$$A^{(h)} = \{(i, j) : i \in N^{(h)}, j \in N^{(h)}\}, \qquad G^{(h)} = \{N^{(h)}, A^{(h)}\}. \tag{16}$$

It is clear that Dantzig's method is valid for *graph growth by subgraphs* for any permutation $\mathcal{P}(N)$.

It would be possible to develop an algorithm for the construction of the rth row of $D^{(h)}$, using Dantzig's method. This would be based on the graph-growth concept and would make a distinction between unlabeled and labeled and scanned nodes. However, this algorithm is likely to remain part of folklore!

VI. CONCLUSION

The results presented show that a unified interpretation of all the different families of methods can be obtained by using graph growths. This approach enables a better analysis of the behavior and complexity of the algorithms to be made. A well-defined method for calculating the SP requires reference to a PQ and to a graph growth. Furthermore, the work to be done at each iteration can be considered as a reoptimization step to get from a SP tree $T^{(h-1)}$ to a new $T^{(h)}$.

I am grateful to Michael Florian, Sang Nguyen and Heinz Spiess for their pertinent criticism and useful suggestions. This work was completed during a visit at the Center of Transportation Research of the University of Montreal and supported by a grant from the National Research Council of Italy.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA (1974).

[2] R. E. Bellman, On a routing problem. *Quart. Appl. Math.* 16 (1958) 87-90.

[3] G. B. Dantzig, All shortest routes in a graph. *Proceeding of the International Symposium on the Theory of Graphs, Rome, Italy, 1966*. Dunod, Paris (1967).

[4] E. V. Denardo and B. L. Fox, Shortest route methods: 1. Reaching, pruning and buckets. *Operations Res.* 27 (1979) 161–186.
[5] R. Dial, Algorithm 360, shortest path forest with topological ordering. *Comm. ACM* 12 (1969) 632–633.
[6] R. Dial, F. Glover, D. Karney, and D. Klingman, A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks* 9 (1979) 215–248.
[7] E. W. Dijkstra, A note on two problems in connection with graphs. *Numer. Math.* 1 (1959) 269–271.
[8] S. E. Dreyfus, An appraisal of some shortest path algorithms. *Operations Res.* 17 (1969) 395–412.
[9] R. W. Floyd, Algorithms 97, shortest path. *Comm. ACM* 5 (1962) 345.
[10] L. J. Ford, Jr., Network flow theory. *Rand Corporation Report P-293* (1956).
[11] G. Gallo, Updating shortest paths in large scale networks. Presented at the International Workshop on Advances in Linear Optimization, Algorithms and Software, Pisa, 1980. To appear in *NETWORKS*.
[12] G. Gallo and S. Pallottino, Shortest path methods: A unifying approach. Presented at the International Workshop on Network Flow Optimization Theory and Practice, Pisa, 1983.
[13] J. Gilsinn and C. Witzgall, A performance comparison of labeling algorithms for calculating shortest path trees. NBS Technical Note 772, Department of Commerce, Washington, DC (1973).
[14] T. C. Hu, Revised matrix algorithms for shortest paths. *SIAM J. Appl. Math.* 15 (1967) 207–218.
[15] D. B. Johnson, A note on Dijkstra's shortest path algorithm. *J. Assoc. Comput. Mach.* 20 (1973) 385–388.
[16] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, MA (1968).
[17] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York (1976).
[18] E. F. Moore, The shortest path through a maze. In *International Symposium on the Theory of Switching 1957*. Harvard University, Cambridge, MA (1959).
[19] S. Pallottino, Adaptation de l'algorithme de D'Esopo–Pape pour la détermination de tous les chemins les plus courts: améliorations et simplifications. Centre de recherche sur les transports, Université de Montréal, Publication No. 136 (1979).
[20] U. Pape, Implementation and efficiency of Moore algorithms for the shortest route problem. *Math. Programming* 7 (1974) 212–222. Algorithm 562, shortest path lengths. *ACM Trans. Math. Software* 6 (1980) 450–455.
[21] M. Pollack and W. Wiebenson, Solution of the shortest-route problem: a review. *Operations Res.* 8 (1960) 224–230.
[22] D. Van Vliet, Improved shortest path algorithms for transport networks. *Transportation Res.* 12 (1978) 7–20.
[23] S. Warshall, A theorem on Boolean matrices. *J. Assoc. Comput. Mach.* 9 (1962) 11–12.
[24] J. W. J. Williams, Algorithms 232: Heapsort. *Comm. ACM* 7 (1964) 347–348.

Received May 10, 1982
Accepted July 28, 1983