THE K SHORTEST PATHS PROBLEM

Ernesto de Queirós Vieira Martins
Marta Margarida Braz Pascoal
José Luis Esteves dos Santos

{*eqvm,marta,zeluis*}*@mat.uc.pt*
Departamento de Matemática
Universidade de Coimbra
PORTUGAL

June 1998

Abstract: The shortest path problem is a classical network programming problem that has been extensively studied. The problem of determining not only the shortest path, but also listing the K shortest paths (for a given integer $K > 1$) is also a classical one but has not been studied so intensively, despite its obvious practical interest.

Two different types of problems are usually considered: the unconstrained and the constrained K shortest paths problem. While in the former no restriction is considered in the definition of a path, in the constrained K shortest paths problem all the paths have to satisfy some condition – for example, to be loopless.

In this paper we are concerned with the unconstrained ranking shortest paths problem. In the first part the problem is viewed as a generalization of the classical shortest path problem; conditions for finiteness and for the optimality principle being satisfied are studied as well as labeling shortest path algorithms are generalized. In the second part the problem is viewed under a different perspective; new algorithms are proposed which compute a super set of the set of the K shortest paths.

Keywords: network, path, path distance, paths ranking, optimality principle, labeling.

---

1 Introduction

The shortest path problem is a classical network programming problem that has been extensively studied (see [6, 7, 8, 9, 12, 22], among hundreds of others possible references). The problem of determining not only the shortest path, but also the second, the third, ..., the $K^{\underline{\text{th}}}$ shortest path (for a given integer $K > 1$) is also a classical one (see the classical papers, [10, 13, 23]) but has not been studied so intensively, despite its obvious practical interest; however more than one hundred of papers can be found in the literature (see the very complete online bibliography due to Eppstein, whose WWW adress is "http://liinwww.ira.uka.de/bibliography/Theory/k-path.html").

Two classes of ranking paths problems can be considered: the unconstrained problem and the constrained one. While in the former no constraint has to be considered on the path definition, in the constrained shortest path problem all the paths have to satisfy some constraint, for example, to be loopless, to be disjoint, etc..

In this paper we are concerned with the unconstrained ranking shortest paths problem. For this problem, the algorithms are divided in two sets: those which are based on the Optimality Principle and those which determine a set which contains the set of the K shortest paths.

The objective of the first part of this paper is to review the problem and to show how it can be considered a generalization of the classical shortest path problem. In fact, althought this generalization seems to be obvious ($K = 1$ for the shortest path problem), it has not been explored for $K > 1$. In the second part of the paper the problem is studied under a different perspective – the set of the K shortest paths forms a tree in the sense of Theory of Graphs, being new algorithms proposed which compute a tree containing at least all the K shortest paths.

The reader is assumed to be familiarized with the problem. Notation is the one used in previous papers, [1, 2, 3, 15, 16, 18]. Despite, some definitions and notation are introduced in the next section.

2 Definitions and Notation

Let $(\mathcal{N}, \mathcal{A})$ denote a given network, where $\mathcal{N} = \{v_1, \ldots, v_n\}$ (or $\mathcal{N} = \{1, \ldots, n\}$ to simplify) is a finite set with n elements called nodes or vertices and $\mathcal{A} = \{a_1, \ldots, a_m\}$ is a finite set with m elements called arcs or edges; moreover, each arc a_k is a pair (i, j) of nodes. When all the m arcs (i, j) are unordered pairs $(\mathcal{N}, \mathcal{A})$ is said to be an undirected network; when all the m arcs (i, j) are ordered pairs $(\mathcal{N}, \mathcal{A})$ is said to be a directed network. In what follows, with no loss of generality and unless something is explicitly said, $(\mathcal{N}, \mathcal{A})$ is a directed network.

Let s and t be two given nodes of $(\mathcal{N}, \mathcal{A})$. A path from s to t in $(\mathcal{N}, \mathcal{A})$ is an alternating sequence of nodes and arcs, of the form $\langle s = v'_1, a'_1, v'_2, \ldots, a'_{r-1}, v'_r = t \rangle$, where:

1. $a'_k \in \mathcal{A}$, for any $k = 1, \ldots, r-1$, and $v'_k \in \mathcal{N}$, for any $k = 1, \ldots, r$;
2. $a'_k = (v'_k, v'_{k+1})$, for any $k = 1, \ldots, r-1$;

Let $c(a_k)$ (or $c_{ij} \in \mathbb{R}$) be a real number associated with arc $a_k = (i, j)$ and let $c(p) = \sum_{(i,j) \in p} c_{ij}$ (or $c(p) = \sum_{p} c_{ij}$), for any path p that can be defined between a pair of nodes of $(\mathcal{N}, \mathcal{A})$, denote the value of path p – or p path distance (or p path cost).

Let $\mathcal{P}_{ij}$ denote the set of paths that can be defined from i to j in $(\mathcal{N}, \mathcal{A})$ ($\mathcal{P}$ will be used for $\mathcal{P}_{st}$). In what follows it is assumed that $\mathcal{P}_{si} \neq \emptyset$ and $\mathcal{P}_{it} \neq \emptyset$, for any node $i \in \mathcal{N}$. Moreover, with no loss of generality, it is assumed that $r \geq 3$, for all the paths that can be defined from s to t in a given network. In fact, generality is not lost, since the given network can be enlarged by the adding of two different nodes, S and T, and two zero distance arcs, (S, s) and (t, T). There is an obvious bijection between $\mathcal{P}_{ST}$ and $\mathcal{P}_{st}$ and there is also no loss of generality when it is assumed that both the initial and terminal nodes are not repeated in a path.

A path from s to t in $(\mathcal{N}, \mathcal{A})$ is a loopless path, if all its nodes are different. A cycle is a path from some node to itself, where all the nodes are different except the first which is also the last one. A cycle C in $(\mathcal{N}, \mathcal{A})$ is said to be a negative cycle *if and only if* $\sum_{C} c_{ij} < 0$, i.e., the sum of its arcs distance is negative; C is said to be an absorvent cycle *if and only if* $\sum_{C} c_{ij} \leq 0$, i.e., the sum of its arcs distance is no positive.

In the K shortest paths problem, for a given network $(\mathcal{N}, \mathcal{A})$ and two nodes s and t ($s \neq t$), it is intended to determine a set $\mathcal{P}^K = \{p_1, \ldots, p_K\} \subset \mathcal{P}$, such that:

1. $c(p_k) \leq c(p_{k+1})$, for any $k \in \{1, \ldots, K-1\}$;
2. $c(p_K) \leq c(p)$, for any $p \in \mathcal{P} - \mathcal{P}^K$;

3. p_k is determined just before p_{k+1}.

Let $p_{ij} \in \mathcal{P}_{ij}$ and $p_{j\ell} \in \mathcal{P}_{j\ell}$ be two paths defined in $(\mathcal{N}, \mathcal{A})$. Since the terminal node of p_{ij} is also the initial node of $p_{j\ell}$, the concatenation of p_{ij} and $p_{j\ell}$, denoted by $p_{ij} \diamond p_{j\ell}$, is defined as the path from i to ℓ in $(\mathcal{N}, \mathcal{A})$, which coincides with p_{ij} from i to j and with $p_{j\ell}$ from j to ℓ.

Let C be a given cycle in $(\mathcal{N}, \mathcal{A})$. For any $k \geq 1$, C^k will denote the k cycle C concatenation $\underbrace{C \diamond \ldots \diamond C}_{k \text{ times}}$, that is, k turns around cycle C.

Let p be some path from s to t in $(\mathcal{N}, \mathcal{A})$ and let C be a given cycle in $(\mathcal{N}, \mathcal{A})$ such that its node i is also a node of p; let p_{si} and p_{it} be two paths in $(\mathcal{N}, \mathcal{A})$ such that $p = p_{si} \diamond p_{it}$. Under these assumptions, $p \diamond C^k$ will denote the path $p_{si} \diamond C^k \diamond p_{it}$, for any $k \geq 0$. (For convenience, $p \diamond C^0$ is also used to denote p).

A path is said to be finite if the number of its nodes (and arcs) is finite.

Theorem 1 – *For any $i \in \mathcal{N}$, let us assume that $\mathcal{P}_{si} \neq \emptyset$ and $\mathcal{P}_{it} \neq \emptyset$. There is a finite shortest path from s to t in $(\mathcal{N}, \mathcal{A})$ if and only if there are no negative cycles in $(\mathcal{N}, \mathcal{A})$.*

Proof – Let us assume the existence of a cycle C in $(\mathcal{N}, \mathcal{A})$ such that $\sum_{C} c_{ij} < 0$ and let i be some node of C. Since, by assumption, $\mathcal{P}_{si} \neq \emptyset$ and $\mathcal{P}_{it} \neq \emptyset$, there is a path $p_{si} \in \mathcal{P}_{si}$ and a path $p_{it} \in \mathcal{P}_{it}$, such that $p = p_{si} \diamond p_{it}$ is a path from s to t. Moreover, since i is a node from path p and from cycle C, it can be concluded that $p(k) = p \diamond C^k$ is a path from s to t for any $k \geq 0$. Let $p_\infty = \lim_{k \to \infty} (p(k))$. So,

$$\begin{aligned}
c(p_\infty) &= c(\lim_{k \to \infty} \{p(k)\}) \\
&= \lim_{k \to \infty} \{c(p(k))\} \\
&= \lim_{k \to \infty} \{c(p \diamond C^k)\} \\
&= \lim_{k \to \infty} \{c(p) + c(C^k)\} \\
&= \lim_{k \to \infty} \{c(p)\} + \lim_{k \to \infty} \{c(C^k)\} \\
&= c(p) + \lim_{k \to \infty} \{k \times c(C)\} \\
&= c(p) + c(C) \times \lim_{k \to \infty} \{k\} \\
&= -\infty.
\end{aligned}$$

That is, p_∞ is a shortest path and it has a non finite number of nodes. Moreover, if q is a finite path from s to t in $(\mathcal{N}, \mathcal{A})$, then $c(q)$ is finite and q is not a shortest path. So, it may be concluded that there is no finite shortest path in $(\mathcal{N}, \mathcal{A})$.

Let us assume now the non existence of negative cycles in $(\mathcal{N}, \mathcal{A})$. In this case, since $\mathcal{P}$ is a non empty set, it is well known that there is a shortest path p which is a loopless path. So, p is a finite path. ∎

A particular K shortest paths problem is said to be finite if there is a finite $K^{\underline{\text{th}}}$ shortest path.

Theorem 2 – *Let us assume that $\mathcal{P}_{si} \neq \emptyset$ and $\mathcal{P}_{it} \neq \emptyset$, for any $i \in \mathcal{N}$. The K shortest paths problem is finite if and only if there are no negative cycles in $(\mathcal{N}, \mathcal{A})$.*

Proof – It follows directly from theorem 1. ∎

In order to assure the finiteness of the K shortest paths problem, it is assumed the existence of no negative cycles in $(\mathcal{N}, \mathcal{A})$ throughout. Moreover, to avoid that the last solutions are of the form $p \diamond C^k$, it is assumed that $c(C) > 0$ for any cycle C in $(\mathcal{N}, \mathcal{A})$, that is, there are no absorvent cycles in $(\mathcal{N}, \mathcal{A})$.

3 Algorithms based on the Optimality Principle

It is well known that shortest path labeling algorithms are supported by the Optimality Principle which asserts that *there is a shortest path formed by shortest sub–paths.* It could be proved that the shortest path problem satisfies the Optimality Principle if and only if *there are no negative cycles in the network.*

This principle is generalized for $K > 1$, in theorem 3.

Theorem 3 – *Let us assume that* $c(C) \geq 0$ *for any cicle* C *in* $(\mathcal{N}, \mathcal{A})$ *and* $\mathcal{P}_{si} \neq \emptyset$, $\mathcal{P}_{it} \neq \emptyset$ *for any* $i \in \mathcal{N}$. *The* $k^{\underline{\text{th}}}$ *shortest path is formed by* $j^{\underline{\text{th}}}$ *shortest paths, for* $j \leq k$.

Proof – Let us assume that the $k^{\underline{\text{th}}}$ shortest path p_k is not formed by $j^{\underline{\text{th}}}$ shortest paths, for $j \leq k$, i.e., $p_k = p_{s\ell} \diamond p^j_{\ell h} \diamond p_{ht}$ and $p^j_{\ell h}$ is a subpath of p_k which is the $j^{\underline{\text{th}}}$ shortest path from ℓ to h and $j > k$. Since $j > k$ we conclude that there are, at least, k different paths $p^i_{\ell h}$ from ℓ to h such that $c(p^i_{\ell h}) < c(p^j_{\ell h})$, for any $1 \leq i \leq k < j$. But $c(p_k) = c(p_{s\ell}) + c(p^j_{\ell h}) + c(p_{ht})$ and $c(p^i_{\ell h}) < c(p^j_{\ell h})$, for any $i \leq k < j$; so, as $c(p^j_{\ell h})$ is finite (theorem 2) we conclude that $c(p_{s\ell}) + c(p^i_{\ell h}) + c(p_{ht}) < c(p_k)$, for any $i \leq k < j$. That is, p_k is not the $k^{\underline{\text{th}}}$ shortest path which contradicts the assumption made. ∎

A class of ranking shortest paths algorithms is supported by theorem 3. Dreyfus, [10], was the first one to propose an algorithm in this class. This algorithm determines not only the K shortest paths between a given pair of nodes but also the K shortest paths from all nodes to t.

Yet in this class of algorithms we can consider the generalization of the labeling algorithms for the shortest path problem. Shier was the first one to propose algorithms in this subclass [19, 20, 21].

Labeling algorithms will be presented in the next subsection.

3.1 Labeling Algorithms

As in shortest path labeling algorithms, their generalizations for the K shortest paths problem are divided into two sets: the label correcting algorithms and the label setting algorithms. While label correcting algorithms determine the set $\mathcal{P}^K = \{p_1, \ldots, p_K\}$ only after all the operations are finished, in the label setting algorithms each path p_k is being determined throughout the computations. That is, while in the label setting algorithms it is well known that a path is the following one in $\mathcal{P}^K$ as soon as a path from s to t is determined, in label correcting algorithms this is known only at the ending of the algorithm.

Labeling algorithms have in common the use of one or more labels assigned to each node. While in the shortest path problem there is a single label assigned to each node i, in the K shortest paths problem we may have a single label or several labels assigned to each node. In the former case the label of node i is formed by three K–tuple: π_i, ξ_i and θ_i. While π^k_i, the $k^{\underline{\text{th}}}$ component of π_i, is the distance of a path from s to node i, the $k^{\underline{\text{th}}}$ component of ξ_i and θ_i, respectively ξ^k_i and θ^k_i, is the node before i in that path and its position in ξ_i, respectively. It is immediate that all the components of π_i are initialized to $+\infty$ for all nodes $i \in \mathcal{N}$ but the first component of π_s which is initialized to zero.

Algorithm 1 is the general form of the K shortest paths labeling algorithm.

From algorithm 1 it can be concluded that the three K–tuple labeling process is not efficient. Among other reasons will be pointed out the following:

1. Let i be some node picked from X in the cycle **while** of the algorithm and let us assume that the label of i is updated later on. So, i is included again in X and when picked from X some calculations can not be executed again for some finite values of π^k_i – those which were *used* before the updating of π_i. To achieve this a *flag* associated with each node i can be used; in this case, the flag of i would be a K–tuple whose $k^{\underline{\text{th}}}$ component is assigned with true if and only if π^k_i has already been previously used.

Algorithm 1 – General form of the labeling K shortest paths algorithm.

/* X — subset of the set of nodes. */

$\{\pi_i^1, \ldots, \pi_i^K\} \longleftarrow \{+\infty, \ldots, +\infty\}$, **for all** node $i \in \mathcal{N}$
$\pi_s^1 \longleftarrow 0$; π_s^1 is *unused*
$X \longleftarrow \{s\}$
while $X \neq \emptyset$
do begin
  $i \longleftarrow$ node of X
  $X \longleftarrow X - \{i\}$
  for each arc $(i,j) \in \mathcal{A}$ **do**:
    for each $k \in \{1, \ldots, K\}$ such that π_i^k is *unused* and finite
    do begin
      π_i^k is *used*
      if $(\pi_i^k + c_{ij} < \max\{\pi_j^1, \ldots, \pi_j^K\})$
      then begin
        $\ell \longleftarrow$ order of $\max\{\pi_j^1, \ldots, \pi_j^K\}$ in $\{\pi_j^1, \ldots, \pi_j^K\}$
        $\pi_j^\ell \longleftarrow \pi_i^k + c_{ij}$; $\xi_j^\ell \longleftarrow i$; $\theta_j^\ell \longleftarrow k$
        π_j^ℓ is *unused*
        if $j \notin X$ **then** $X \longleftarrow X \cup \{j\}$
      end
    end
end
Find the K shortest paths

---

2. It may happen that some components of some labels remain $+\infty$ throughout the algorithm. This will happen since the number of paths from s to some node i is less than K. In this case, it is obvious that the memory space is not being efficiently used.

3. To assure the algorithm finiteness, the maximum component of the distance label of a node j has to be computed each time that a node i is picked on X and for each finite π_i^k, which represents a really large amount of computational effort.

Computationally it is really difficult to deal with the three K–tuple labels used in the general form of the labeling K shortest paths algorithm. Moreover, it may happen that K is unknown in advance; for example, if it is intended to determine the shortest path which satisfies some predefined conditions. In this case, the three K–tuple labels can not be used. So, a different version of this algorithm can be designed where each node is assigned with several labels. Basically, each finite component of the distance label of node i is considered as a label of node i itself.

From now on, this form will be used in the description of labeling K shortest paths algorithms. To simplify the notation (and because it results computationally more efficient), the set of nodes will be enlarged in such a way that each node is assigned with a single label. A function $h : \mathbb{N} \longrightarrow \mathcal{N}$ has to be defined in order to make possible to assign a single label to some natural number, i.e., to some well determined node. Since some nodes may be assigned with more than a single label, the h function has no inverse; however and to make possible the determination of all the natural numbers which correspond to the same node we will define $h^{-1}(i) = \{k \in \mathbb{N} \mid h(k) = i\}$. Note that the set $h^{-1}(i)$ is easily computed for any $i \in \mathcal{N}$ once it is viewed as a linked list.

Algorithm 2 is the general form of the K shortest paths labeling algorithm, where the set of nodes is enlarged in order that each node is assigned with a single label. Note that the enlarged set of nodes

has at most nK elements.

We must notice that the single label algorithm does not work for unknown values of K. Furthermore, the determination of $\max\{\pi_x \mid x \in h^{-1}(j)\}$ for each arc (i,j) and each time that some k is picked on X assures that the number of labels assigned to each node is no greater than K.

3.1.1 Label Correcting Algorithms

The general form of this set of algorithms is that one which has been stated for the general case, that is, the form described in algorithm 1 (or that one described in algorithm 2). When there is no freedom to choose the node i in X (or the element k in X) and no freedom in the adding of a node j to X (or of an element *elm* to X) a specific form is designed. For example, X may be implemented as a **FIFO** linked list which results in the generalization of the Bellman–Ford–Moore form (**BFM** form) for the shortest path problem, [4, 5, 17], which is described in algorithm 3. In this algorithm, each node is assigned with a single label.

This form of the label correcting algorithm will be exemplified with the network represented in figure 1. For this purpose, the set of arcs will be considered arranged in such a way that $\mathcal{A} = \{a_1, \ldots, a_m\}$ and $a_k = (i,j) < a_\ell = (i',j')$ if and only if $i < i'$ or $i = i'$ and $j < j'$. Furthermore, in order to increase the readability of the example, the function h so as h^{-1}, *elm* and *count* will not be used. Instead, a node i will be primed with so many primes as the number of paths that were determined from s to i.

The ranking will be executed for $K = 2$ paths from $s = 5$ to $t = 1$. The obtained results are depicted in figure 2 (the first seven steps) and in figure 3 (the optimal solution with all the determined labels). Each circle is associated with a well determined path wich is defined from $s = 5$ until the node with primes within the circle. Closed to each circle we find a number which represents the distance of this path; the ξ is represented by the arrows.

Dashed circles represent labels that had been computed and later removed. We must notice the dashed subtree rooted at label $2'$; all the labels in this sutree were removed only after being picked from X and being used to determine wrong labels.

From the example it is immediate to conclude that the **BFM** form of the label correcting algorithm is not really efficient; we can even say that it works like a *brute force* method. Notice that in the brute force method all the possibilities are tested which implies the nonfiniteness of the algorithm unless the number of paths is finite (such as in acyclic networks). So, to assure finiteness for the general case, we use the Principle of Optimality considering the K shortest paths from s to any node i, such as in label correcting algorithms.

We must also notice that the theoretical complexity of this algorithm is not polinomial for a worst case analysis since it is not possible to establish a polinomial upper bound for the number of labels which are computed for each node.

We must notice that both forms of the algorithm, i.e., the form where each node is assigned with a single label and the form where each node is assigned with K labels, are coincident when $K = 1$. Moreover, in this case, algorithm 3 is also the Bellman–Ford–Moore form of the label correcting algorithm for the single shortest path problem.

Other forms of the label correcting algorithm could be designed, either with a K–tuple label assigned to each node or manipulating X in different ways. However, it appears that its either practical or theoretical efficiency is not improved.

It must be also pointed out that the set $\mathcal{P}^K$ is determined after the execution of the algorithm; moreover label correcting algorithms determine not only the K shortest paths from s to t, but also the K shortest paths from s to i, for all $i \in \mathcal{N}$.

Algorithm 2 – General form of the single label K shortest paths algorithm.

```
/*
        X        —   subset of N*, the enlarged set of nodes.
        count_i  —   number of paths that were determined from s to i.
        elm      —   number of elements of N*.
                                                                    */
count_i ⟵ 0, for all i ∈ N
elm ⟵ 1
h(elm) ⟵ s; h^{-1}(s) ⟵ {elm}; π_elm ⟵ 0
X ⟵ {elm}
while (X ≠ ∅)
do begin
        k ⟵ some element of X
        X ⟵ X − {k}
        i ⟵ h(k)
        for each arc (i, j) ∈ A
        do begin
                if (count_j = K)
                then begin
                        ℓ ⟵ element of h^{-1}(j) for which π_ℓ = max{π_x | x ∈ h^{-1}(j)}
                        if (π_k + c_ij < π_ℓ)
                        then begin
                                π_ℓ ⟵ π_k + c_ij; ξ_ℓ ⟵ k
                                if ℓ ∉ X then X ⟵ X ∪ {ℓ}
                             end
                     end
                else begin          /* (count_j < K) */
                        elm ⟵ elm + 1
                        π_elm ⟵ π_k + c_ij; ξ_elm ⟵ k
                        count_j ⟵ count_j + 1
                        h(elm) ⟵ j; h^{-1}(j) ⟵ h^{-1}(j) ∪ {elm}
                        X ⟵ X ∪ {elm}
                     end
          end
  end
Find the K shortest paths
```

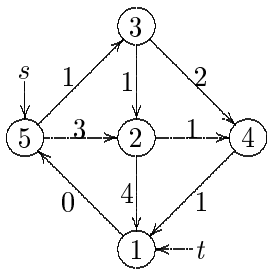

Figure 1: **Network to exemplify algorithms**

Algorithm 3 – Generalization of the Bellman–Ford–Moore form with single labels.

```
/*
      X        —  a sequence of nodes of N*, the enlarged set of nodes.
      count_i  —  number of paths that were determined from s to i.
      elm      —  number of elements of N*.
                                                                    */
count_i ⟵ 0, for all i ∈ N
elm ⟵ 1
h(elm) ⟵ s; h^{-1}(s) ⟵ {elm}; π_elm ⟵ 0
X ⟵ {elm}
while X ≠ ∅
do begin
      k ⟵ first element of X
      X ⟵ X − {k}
      i ⟵ h(k)
      for each arc (i, j) ∈ A
      do begin
            if (count_j = K)
            then begin
                  ℓ ⟵ element of h^{-1}(j) for which π_ℓ = max{π_x | x ∈ h^{-1}(j)}
                  if (π_k + c_ij < π_ℓ)
                  then begin
                        if ℓ ∈ X then X ⟵ X − {ℓ}
                        π_ℓ ⟵ π_k + c_ij; ξ_ℓ ⟵ k
                        X ⟵ X ∪ {ℓ} and ℓ is the last element of X
                       end
                 end
            else begin        /* (count_j < K) */
                  elm ⟵ elm + 1
                  π_elm ⟵ π_k + c_ij; ξ_elm ⟵ k
                  count_j ⟵ count_j + 1
                  h(elm) ⟵ j; h^{-1}(j) ⟵ h^{-1}(j) ∪ {elm}
                  X ⟵ X ∪ {elm} and elm is the last element of X
                 end
       end
  end
Find the K shortest paths
```

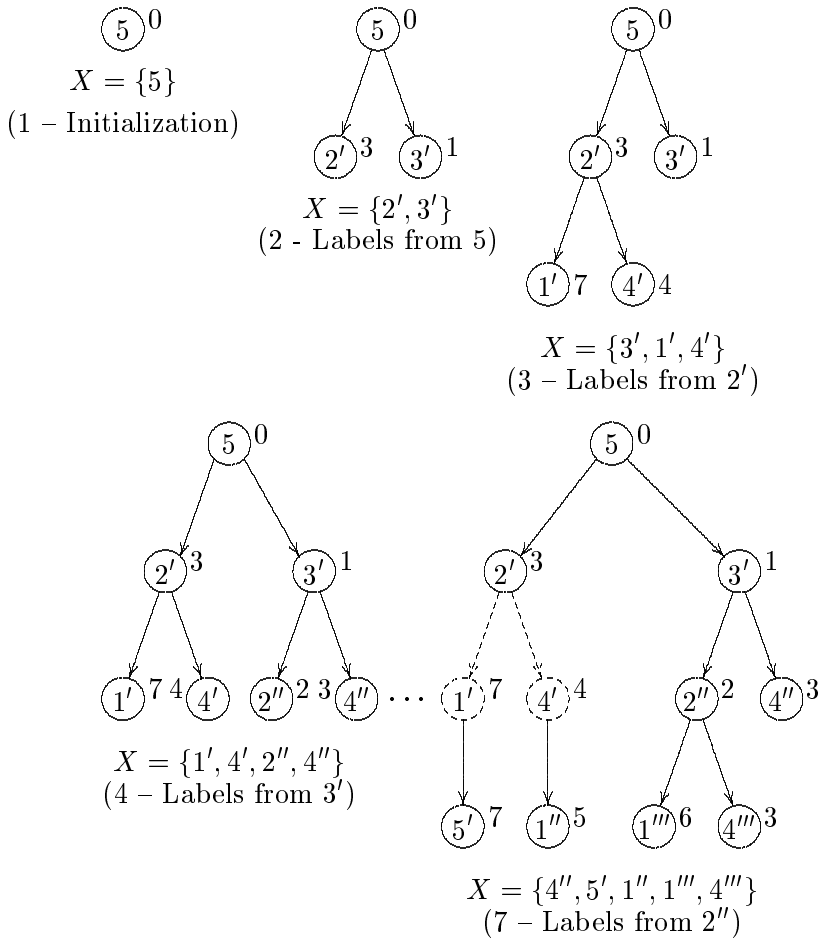

Figure 2: **The first seven steps of the BFM generalization form**

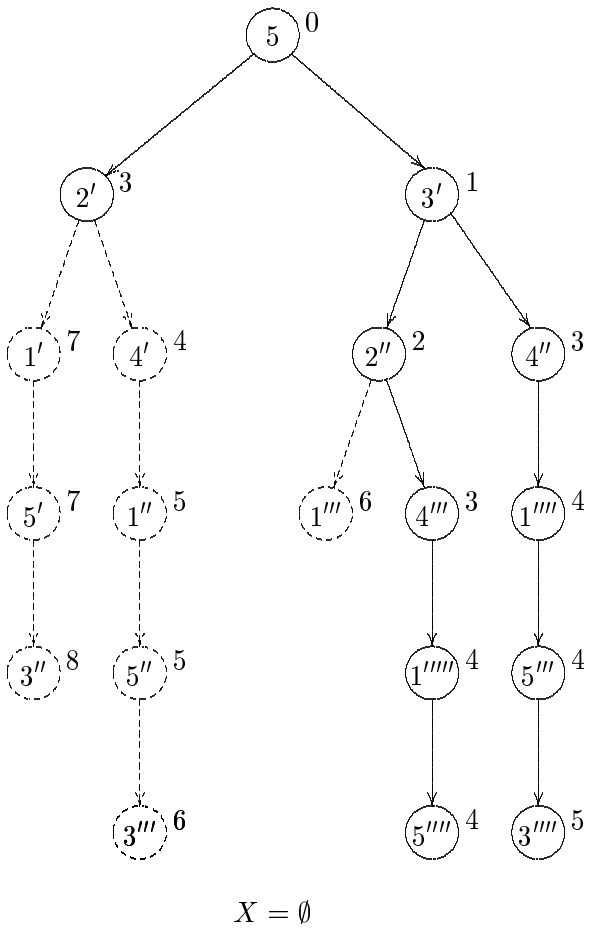

Figure 3: **Labels obtained with the BFM form to determine the $K = 2$ shortest paths**

In the class of the labeling correcting algorithms are included those proposed by Shier, [19, 20, 21].

3.1.2 Label Setting Algorithms

The efficiency of label correcting algorithms can be easily improved. In fact we may think to manipulate X in such a way that the label of the element picked on X is permanent, such as in Dijkstra's algorithm for the shortest path problem, [9]. So, let us assume, with no loss of generality, that $c_{ij} \geq 0$, for all $(i,j) \in \mathcal{A}$. Once for $K = 1$ the algorithm 4 is the classical Dijkstra's algorithm for the single shortest path problem, it is immediate that algorithm 4 is its generalization for the K shortest paths problem; so, it is a label setting algorithm. Notice that $\mathcal{P}^K$ is being computed along

---

Algorithm 4 – Generalization of Dijkstra's algorithm, with single labels.

```
/*
        X      —   subset of N*, the enlarged set of nodes.
   count_i     —   number of paths that were determined from s to i.
      elm      —   number of elements of N*.
                                                                  */
count_i ⟵ 0, for all i ∈ N
elm ⟵ 1
h(elm) ⟵ s; h^{-1}(s) ⟵ {elm}; π_elm ⟵ 0
X ⟵ {elm}
P^K ⟵ ∅
While (count_t < K) and (X ≠ ∅)
do begin
        k ⟵ element of X such that π_k ≤ π_x, for any x ∈ X
        X ⟵ X − {k}
        i ⟵ h(k)
        count_i ⟵ count_i + 1
        if (i = t)
        then begin
                p ⟵ path from 1 to k
                P^K ⟵ P^K ∪ {h(p)}        /*  p_{count_t} ⟵ h(p)  */
             end
        if (count_i ≤ K)
        then begin
                for each arc (i,j) ∈ A
                do begin
                        elm ⟵ elm + 1
                        π_elm ⟵ π_k + c_ij; ξ_elm ⟵ k
                        h(elm) ⟵ j; h^{-1}(j) ⟵ h^{-1}(j) ∪ {elm}
                        X ⟵ X ∪ {elm}
                   end
             end
  end
```

---

the execution of the algorithm. Moreover, whenever an element k is picked on X for which $h(k) = t$ a new path is added to $\mathcal{P}^K$.

A different form of the algorithm could be designed. Namely, keeping in X only the K shortest paths from s to i, for any $i \in \mathcal{N}$. However, this form would be less efficient in terms of the execution time since keeping the K shortest paths from s to all $i \in \mathcal{N}$ is a very expansive task (at least for

significant values of K); the described form is more efficient with respect to the execution time but more labels are assigned which implies less efficiency in terms of the necessary memory space.

The theoretical computational complexity of this form can be established, considering a worst case analysis. So let us assume that p_1, the shortest path from s to t, is an hamiltonean path, i.e., it has $n-1$ arcs and it is a loopless path. Moreover let us assume that there is the arc $(t,s) \in \mathcal{A}$ such that $c_{ts} = 0$; that is, p_k, the $k^{\underline{\text{th}}}$ shortest path, is $\underbrace{p_1 \diamond (t,s) \diamond p_1 \diamond \ldots \diamond (t,s) \diamond p_1}_{k \text{ times}}$. So each node is assigned with, at most, K permanent labels. The number of elements that are generated from each permanent label of node i, is exactly the number of arcs whose tail node is i; that is, X has, at most, Km elements. So the required memory space is $\mathcal{O}(Km)$; moreover, using a suitable data structure to represent X, the time complexity is also $\mathcal{O}(Km)$.

To exemplify the algorithm, the problem depicted in figure 1 will be solved for $K = 2$ (see figure 4).

We must notice that either Dreyfus' algorithm, [10], or **MS** algorithm, [16], and its previous versions, [1, 2, 3, 14], are also label setting algorithms for the shortest paths ranking problem.

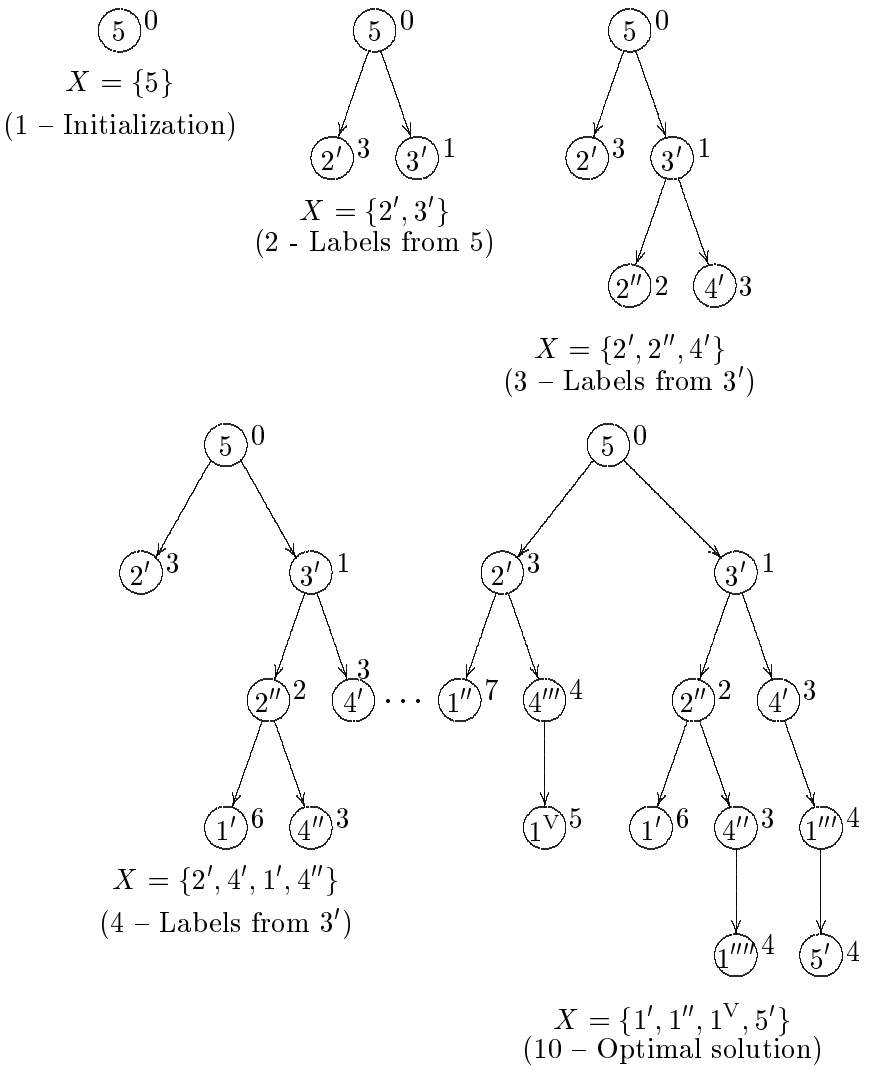

Figure 4: **Ten steps of the Dijkstra's generalization form**

4 The tree of the K shortest paths

Let $\mathcal{P}^K = \{p_1, \ldots, p_K\}$ be the set of the K shortest paths and let us consider algorithm 5 exemplified in figure 5 for $\mathcal{P}_3 = \{p_1, p_2, p_3\}$ where $p_1 = \{1, (1,4), 4, (4,6), 6\}$, $p_2 = \{1, (1,4), 4, (4,1), 1, (1,6), 6\}$ and $p_3 = \{1, (1,6), 6\}$.

Algorithm 5 – Constructing a pseudo–tree, the tree of the K shortest paths.

/* $\mathcal{T}$ — Pseudo–tree of the K shortest paths. */

$\mathcal{T} \leftarrow \emptyset$
$k \leftarrow 1$
While $k \leq K$
do begin
  $arc \leftarrow$ first arc of p_k
  $head \leftarrow$ head node of arc
  $p \leftarrow$ subpath of p_k from s to $head$
  While $p \in \mathcal{T}$
  do begin
    $arc \leftarrow$ arc following arc in p_k
    $head \leftarrow$ head node of arc
    $p \leftarrow$ subpath of p_k from s to $head$
  end
  $v_k \leftarrow$ tail node of arc
  $p^k_{v_k t} \leftarrow$ subpath of p_k from v_k to t
  $p^k_{s v_k} \leftarrow$ subpath of p_k from s to v_k
  $\mathcal{T} \leftarrow \mathcal{T} \cup p^k_{v_k t}$ /* In such a way that $p_k = p^k_{s v_k} \diamond p^k_{v_k t}$ is a path of $\mathcal{T}$ */
  $k \leftarrow k + 1$
end

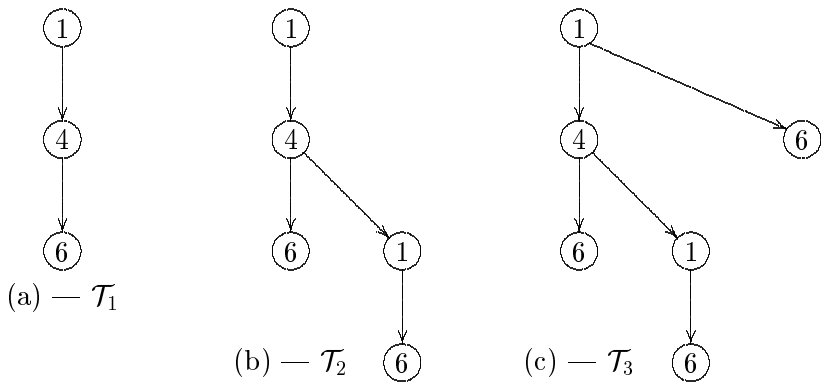

Figure 5: **The trees of shortest paths**

For $k = 1$ the following values are sucessively obtained: $arc \leftarrow (1,4)$, $head \leftarrow 4$, $p \leftarrow \{1, (1,4), 4\}$; since $\mathcal{T}$ is empty it can be concluded that $p \notin \mathcal{T}$ and $v_1 \leftarrow 1$, $p^1_{v_1 6} \leftarrow p_1$, $p^1_{1 v_1} \leftarrow \emptyset$, $\mathcal{T} \leftarrow p_1, k \leftarrow 2$ – see figure 5(a).

For $k = 2$, since $p_2 = \{1, (1,4), 4, (4,1), 1, (1,6), 6\}$, $arc \leftarrow (1,4)$, $head \leftarrow 4$ and $p \leftarrow \{1, (1,4), 4\}$; as $\{1, (1,4), 4\} \in \mathcal{T}$ then $arc \leftarrow (4,1)$, $head \leftarrow 1$ and $p \leftarrow \{1, (1,4), 4, (4,1), 1\}$ which is not a path of

$\mathcal{T}$. That is, $v_2 \leftarrow 4$, $p^2_{46} \leftarrow \{4,(4,1),1,(1,6),6\}$ and $p^2_{14} \leftarrow \{1,(1,4),4\}$ is added to $\mathcal{T}$ in such a way that p_2 is a path of $\mathcal{T}$ – see figure 5(b).

Similarly the algorithm would continue until $k = 3$ with $arc \leftarrow (1,6)$, $head \leftarrow 6$ and $p \leftarrow \{1,(1,6),6\}$; since p is not a path of $\mathcal{T}$ it is added to the tree – see figure 5(c).

Theorem 4 is immediate from algorithm 5.

Theorem 4 – *The set $\mathcal{P}^k = \{p_1,\ldots,p_k\}$ of the k shortest paths forms a pseudo-tree, for $k \geq 1$.*

For any $k \in \{1,\ldots,K\}$, node v_k determined by algorithm 5 is called the deviation node of p_k; the arc of p_k whose tail node is v_k is called deviation arc of p_k and $p^k_{v_k t}$, the subpath of p_k from v_k to t, is the deviation path of p_k.

From algorithm 5 it is also immediate that nodes are repeated in $\mathcal{T}$, but we will consider all nodes as being different – see section 3.1; so, $\mathcal{T}$ is a pseudo-tree – called the tree of the K shortest paths. The objective of any algorithm for ranking the K shortest paths is to construct $\mathcal{T}$; to achieve this, a pseudo-tree which contains $\mathcal{T}$ is usually constructed. All the algorithms in the class, which are studied in this section, use a set of candidates to p_k – the $k^{\underline{\text{th}}}$ shortest path – which is the shortest in this set. They are supported by theorems 4 and 5.

Theorem 5 – *For $k \geq 1$, let $\mathcal{T}_k$ denote the tree of the k shortest paths and let $p^{k+1}_{v_{k+1}t}$ be the deviation path of p_{k+1}. Then $p^{k+1}_{v_{k+1}t}$ is the shortest path from v_{k+1} to t whose first arc is not an arc of $\mathcal{T}_k$ with v_{k+1} as tail node.*

Proof – Let us assume that $p^{k+1}_{v_{k+1}t}$, the deviation path of p_{k+1}, is not the shortest path from v_{k+1} to t whose first arc is not an arc of $\mathcal{T}_k$ with v_{k+1} as tail node. That is, there is a path q from v_{k+1} to t whose first arc is not an arc of $\mathcal{T}_k$ with v_{k+1} as tail node, such that $\sum_q c_{ij} < \sum_{p^{k+1}_{v_{k+1}t}} c_{ij}$. Since v_{k+1} is the tail node of the first arc of q and this arc is not an arc of $\mathcal{T}_k$, it can be concluded that $q \notin \mathcal{T}_k$. That is, p_{k+1} is not the $(k+1)^{\underline{\text{th}}}$ shortest path which contradicts the hypothesis. Since paths from v_{k+1} to t whose first arc is an arc of $\mathcal{T}_k$ with v_{k+1} as tail node have been already considered, the theorem follows. ∎

4.1 Generalization of Yen's algorithm

For any node $v \in \mathcal{N}$ let $\mathcal{A}(v)$ denote the set of arcs of $(\mathcal{N},\mathcal{A})$ whose tail node is v; for any $k \geq 1$, let $\mathcal{A}_{\mathcal{T}_k}(v)$ denote the set of arcs of $\mathcal{T}_k$ whose tail node is v, a node of $p^k_{v_k t}$ – the deviation path of p_k.

In the algorithm to be described in this section, a set X of paths candidates to the following shortest one is used. Obviously, X is initialized with the shortest path p_1.

In the $k^{\underline{\text{th}}}$ step of the algorithm, the $k^{\underline{\text{th}}}$ shortest path is picked out from X and some new candidates to the $(k+1)^{\underline{\text{th}}}$ shortest path are generated. Since p_k coincides with some p_ℓ, for $1 \leq \ell < k$, from the initial node s until the deviation node v_k of p_k, all nodes until v_k were already considered in the attempt of generating new candidates. So, from all the nodes v of $p^k_{v_k t}$, the shortest path $p^\star_{vt}$, from v to t whose first arc is not an arc of $\mathcal{A}_{\mathcal{T}_k}(v)$, will be computed. From theorem 5, $p^k_{sv} \diamond p^\star_{vt}$ is a new candidate to p_{k+1}. $p^\star_{vt}$ can be easily determined since $\mathcal{T}^\star_t$, the tree of the shortest paths from any node $i \in \mathcal{N} - \{t\}$, has been previously computed. In fact, the arc $(v,x) \in \mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$ has to be determined in such a way that $c_{vx} + c(p^\star_{xt})$ is minimized, where $p^\star_{xt}$ stands for the shortest path from x to t, that is, the path which is determined in $\mathcal{T}^\star_t$ from x to t. It must be noticed that $\mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$ can be the empty set, which means that all deviations paths for node v were already determined. Moreover, the deviation node of some path p_k, for $k \geq 1$ can be the initial node itself, which implies that all nodes of p_k has to be considered in the attempt for generating candidates to p_{k+1}.

Algorithm 6 – Generalization of Yen's algorithm

/* X — Set of candidates to the following shortest path. */

Compute $\mathcal{T}_t^\star$
$p_1 \leftarrow$ shortest path from s to t /* p_1 is a path of $\mathcal{T}_t^\star$ */
$k \leftarrow 1$
$X \leftarrow \{p_k\}$
$\mathcal{T}_k \leftarrow \{p_k\}$
While $k < K$ **and** $X \neq \emptyset$
do begin
  $X \leftarrow X - \{p_k\}$
  $v_k \leftarrow$ deviation node of p_k
  for each node $v \in p^k_{v_k t}$
  do begin
    if $\mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v) \neq \emptyset$
    then begin
      Compute the arc (v, x) such that $c_{vx} + c(p^\star_{xt})$ is minimized over $\mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$
        /* $p^\star_{xt}$ is a path of $\mathcal{T}_t^\star$ */
      $q \longleftarrow p^k_{sv} \diamond \{v, (v, x), x\} \diamond p^\star_{xt}$
      $X \longleftarrow X \cup \{q\}$
      $q_{vt} \longleftarrow \{v, (v, x), x\} \diamond p^\star_{xt}$
      $\mathcal{T}_k \longleftarrow \mathcal{T}_k \cup \{q_{vt}\}$ /* In such a way that $p^k_{sv} \diamond q_{vt}$ is a path of $\mathcal{T}_k$ */
    end
  end
  $k \leftarrow k + 1$
  $p_k \leftarrow$ shortest path in X
end

This algorithm works similarly to Yen's algorithm for ranking loopless paths, [23]. The only difference consists in the definition of path; that is, while in Yen's algorithm only loopless paths are computed, in the proposed algorithm paths may eventually have loops. That is the reason why the algorithm is considered a generalization of Yen's algorithm.

It must be also noticed that $\mathcal{T}_t^\star$ can be easily determined with some labeling algorithm for the classical shortest path problem being enough to reverse the orientation of all the arcs and to consider t as the initial node.

The algorithm is exemplified with the network in figure 6, whose tree of the shortest paths from $i \in \mathcal{N} - \{t\}$ to t is represented in figure 7.

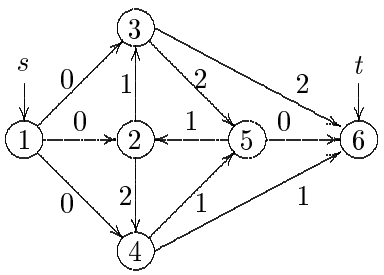

Figure 6: **Network to exemplify algorithms**

So, $p_1 = \{1, (1, 4), 4, (4, 6), 6\}$ is the shortest path from s to t and p_1 is a path of $\mathcal{T}_t^\star$; moreover, v_1,

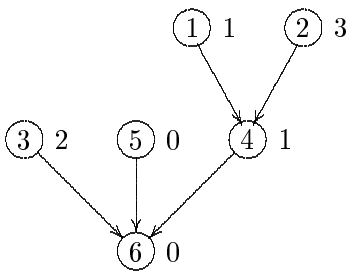

Figure 7: $\mathcal{T}_t^\star$ **for the network in figure 6**

its deviation node, is the initial node $s = 1$. That is, $X = \{p_1\}$ and $\mathcal{T}_1$ is path p_1 itself – see figure 8. Path q_2 is generated from the deviation node of p_1 and added to $\mathcal{T}_1$. In fact, $v_1 = 1$ is the tail node of arcs $(1,2), (1,3)$ and $(1,4)$; since $(1,4)$ is an arc of $\mathcal{T}_1$ with $v_1 = 1$ as tail node, the minimum of $\{c_{12} + c(p_{26}^\star), c_{13} + c(p_{36}^\star)\}$ has to be computed. That is, the minimum of $\{0+3, 0+2\}$ which is attained for $\{c_{13} + c(p_{36}^\star)\}$ – so, $\{1,(1,3),3\} \diamond p_{36}^\star$ is added to $\mathcal{T}_1$.

Similarly for nodes 4 and 6, to obtain $\mathcal{T}_2$ – see figure 8. Notice that no path is added to $\mathcal{T}_1$ from node 6 since $\mathcal{A}(6) = \emptyset$ (which implies that $\mathcal{A}_\mathcal{T}(6) = \emptyset$, for any tree $\mathcal{T}$).

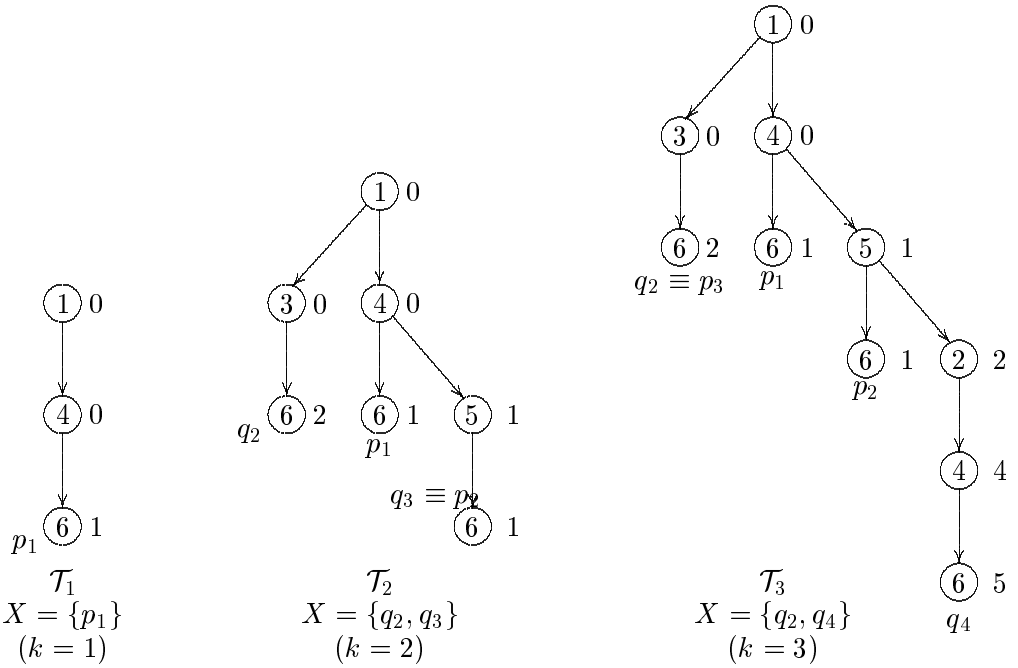

Figure 8: **The first three steps of the generalization of Yen's algorithm**

The computational complexity of Yen's generalization algorithm is $\mathcal{O}(Km)$, after the determination of $\mathcal{T}_t^\star$ and when the worst case analysis is considered. In fact, in the worst case, no more than n different nodes has to be considered after the deviation node, when new candidate paths are being added to the set X. That is, for each k all the set of arcs are analised in the worst case.

4.2 Reduced Costs

Let $\mathcal{T}_t$ be some tree rooted at t, that is, a tree where there is a (unique) path from any node $i \in \mathcal{N} - \{t\}$ to t; for any node $i \in \mathcal{N}$, let $\pi_i(\mathcal{T}_t)$ denote the cost of $p_{it}(\mathcal{T}_t)$ – the path from i to t in

$\mathcal{T}_t$. For any arc $(i,j) \in \mathcal{A}$ let $\bar{c}_{ij} = \pi_j(\mathcal{T}_t) - \pi_i(\mathcal{T}_t) + c_{ij}$ be the reduced cost of arc $(i,j) \in \mathcal{A}$ associated with $\mathcal{T}_t$.

Theorem 6 plays an important role in ranking paths algorithms.

Theorem 6 – *For any two nodes $x, y \in \mathcal{N}$ such that $x \neq y$, let $p, q \in \mathcal{P}_{xy}$ be two paths from x to y. For any tree rooted at t, $\mathcal{T}_t$:*

1 - $c(p) = \sum_p c_{ij} = \sum_q c_{ij} = c(q)$ *if and only if* $\bar{c}(p) = \sum_p \bar{c}_{ij} = \sum_q \bar{c}_{ij} = \bar{c}(q)$.

2 - $c(p) = \sum_p c_{ij} < \sum_q c_{ij} = c(q)$ *if and only if* $\bar{c}(p) = \sum_p \bar{c}_{ij} < \sum_q \bar{c}_{ij} = \bar{c}(q)$.

Proof – The proof follows immediate from the definition of reduced cost. In fact, from $\bar{c}(p) = \sum_p \bar{c}_{ij} = \sum_q \bar{c}_{ij} = \bar{c}(q)$ it can successively be concluded:

$$\sum_p (\pi_j(\mathcal{T}_t) - \pi_i(\mathcal{T}_t) + c_{ij}) = \sum_q (\pi_j(\mathcal{T}_t) - \pi_i(\mathcal{T}_t) + c_{ij}),$$

$$\pi_y(\mathcal{T}_t) - \pi_x(\mathcal{T}_t) + \sum_p c_{ij} = \pi_y(\mathcal{T}_t) - \pi_x(\mathcal{T}_t) + \sum_q c_{ij},$$

$$\sum_p c_{ij} = \sum_q c_{ij}.$$

That is, $c(p) = c(q)$.

The proof is similar for statement 2. ∎

From theorem 6 it can be concluded that ranking paths may be achieved either using the cost $c(p_k)$ (and c_{ij}, for every arc $(i,j) \in \mathcal{A}$) or the reduced cost $\bar{c}(p_k)$ (and $\bar{c}_{ij}$, for every arc $(i,j) \in \mathcal{A}$). The following theorem show the advantage of using reduced costs.

Theorem 7 – *Let $\mathcal{T}_t^\star$ be the shortest tree rooted at t, that is, the tree of shortest paths from every node $i \in \mathcal{N} - \{t\}$ to t. Then, $\bar{c}_{ij} \geq 0$ for every arc $(i,j) \in \mathcal{A}$; moreover, $\bar{c}_{ij} = 0$ for every arc $(i,j) \in \mathcal{A} \cap \mathcal{T}_t^\star$, that is, for any arc in the shortest tree rooted at t.*

Proof – Let $(i,j) \in \mathcal{A}$ be some arc in $(\mathcal{N}, \mathcal{A})$ and let us assume that $\bar{c}_{ij} < 0$. Then, by definition, $\bar{c}_{ij} = \pi_j - \pi_i + c_{ij}$ where π_k stands for $\pi_k(\mathcal{T}_t^\star)$, for any node $k \in \mathcal{N}$. It can be concluded that $\{i,(i,j),j\} \diamond p_{jt}^\star$ is a path from i to t and $c(\{i,(i,j),j\} \diamond p_{jt}^\star) = c_{ij} + c(p_{jt}^\star)$; that is, $c(\{i,(i,j),j\} \diamond p_{jt}^\star) = c_{ij} + \pi_j$. Since $\bar{c}_{ij} = \pi_j - \pi_i + c_{ij} < 0$ then $c_{ij} + \pi_j < \pi_i$, that is, there is a path from i to t whose cost is less than the cost of the shortest path which is impossible. So, $\bar{c}_{ij} \geq 0$. If $(i,j) \in \mathcal{A}$, π_i is the cost of the shortest path $p_{it}^\star$ from i to t which is determined in $\mathcal{T}_t^\star$ and $p_{it}^\star = \{i,(i,j),j\} \diamond p_{jt}^\star$. That is, $\pi_i = c_{ij} + \pi_j$ and $\pi_j - \pi_i + c_{ij} = \bar{c}_{ij} = 0$. ∎

The advantage of using the reduced costs associated to the shortest tree rooted at t in ranking shortest paths algorithms is immediate from theorem 7. For instance, in the generalization of Yen's algorithm more efficiency is achieved in the determination of the arc $(v,x) \in \mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$ which minimizes $\bar{c}_{vx} + \bar{c}(p_{xt}^\star)$, since $p_{xt}^\star$ is a path of $\mathcal{T}_t^\star$ and $\bar{c}(p_{xt}^\star) = 0$ for any node $x \in \mathcal{N}$. That is, in the $k^{\underline{\text{th}}}$ step, it is enough to determine the arc $(v,x) \in \mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$ which minimizes $\bar{c}_{vx}$, for each node v in the deviation path of p_k.

Eppstein was the first to take advantage of using reduced costs associated to the shortest tree rooted at t, [11].

In next section an algorithm is presented that can be viewed as an improvement to the Yen's generalization algorithm.

$(i,j) \in \mathcal{A}$	π_i	π_j	c_{ij}	$\bar{c}_{ij} = \pi_j - \pi_i + c_{ij}$
$(1,2)$	1	3	0	2
$(1,3)$	1	2	0	1
$(1,4)$	1	1	0	0
$(2,3)$	3	2	1	0
$(2,4)$	3	1	2	0
$(3,5)$	2	0	2	0
$(3,6)$	2	0	2	0
$(4,5)$	1	0	1	0
$(4,6)$	1	0	1	0
$(5,2)$	0	3	1	4
$(5,6)$	0	0	0	0

Table 1: Reduced costs associated to the shortest tree rooted at t of figure 7

4.3 MPS algorithm

In section 4.2 it was shown how the efficiency of Yen's generalization algorithm can be improved using reduced costs. However, more efficiency can yet be achieved when a suitable data structure is used to define $(\mathcal{N}, \mathcal{A})$. In fact, in section 4.2 it was proved that, in the $k^{\underline{\text{th}}}$ step of Yen's generalization algorithm, it is enough to determine the arc $(v,x) \in \mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$ which minimizes $\bar{c}_{vx}$, for each node v in the deviation path of p_k. If the set of arcs is grouped for each tail node, and for each set $\mathcal{A}(v)$ arcs are sorted by its reduced costs, the determination of arc $(v,x) \in \mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$ which minimizes $\bar{c}_{vx}$ is immediate – (v,x) it is the first arc in the set $\mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$.

In the next subsection a data structure for representing $(\mathcal{N}, \mathcal{A})$ is defined which allows more efficiency for the algorithm.

4.3.1 Sorted forward star form

Let us arrange $\mathcal{A} = \{a_1, \ldots, a_m\}$ in such a way that $k < \ell$ if and only if the tail node of a_k is not greater than the tail node of a_ℓ; that is, $\mathcal{A}$ is arranged in such a way that $\mathcal{A} = \mathcal{A}(1) \cup \ldots \cup \mathcal{A}(n)$, where n is the number of nodes of $(\mathcal{N}, \mathcal{A})$. Moreover, for every $k \in \{1, \ldots, n\}$, the set $\mathcal{A}(k)$ is sorted by the reduced cost of its arcs; that is, if $\mathcal{A}(k) = \{\hat{a}_1, \ldots, \hat{a}_\ell\}$ then $\bar{c}(\hat{a}_1) \leq \ldots \leq \bar{c}(\hat{a}_\ell)$.

Notice that rearranging $\mathcal{A}$ in the sorted forward star form can be done with $\mathcal{O}(m \log m)$ operations.

More details about the sorted forward star form can be found in [8].

4.3.2 The algorithm

MPS algorithm is stated in algorithm 7 and exemplified with the network in figure 6.

The reduced costs associated to the shortest tree rooted at t – see figure 7 – are shown in table 1.

According to these reduced costs and after rearranging $(\mathcal{N}, \mathcal{A})$ in the sorted forward star form:

$\mathcal{A}(1) = \{(1,4),(1,3),(1,2)\}$, $\mathcal{A}(2) = \{(2,3),(2,4)\}$, $\mathcal{A}(3) = \{(3,5),(3,6)\}$,
$\mathcal{A}(4) = \{(4,5),(4,6)\}$, $\mathcal{A}(5) = \{(5,6),(5,2)\}$ and $\mathcal{A}(6) = \emptyset$.

From now on, all steps of MPS algorithm are coincident with those of Yen's generalization algorithm. However, the determination of new candidate paths results easier. Notice, for instance, that for generating candidates from path p_2, nodes 4, 5 and 6 have to be scanned; while for node $v = 4$ $\mathcal{A}(4) - \mathcal{A}_{\mathcal{T}_2}(4) = \{(4,5),(4,6)\} - \{(4,5),(4,6)\} = \emptyset$, the first arc of $\mathcal{A}(5) - \mathcal{A}_{\mathcal{T}_2}(5)$ is arc $(5,2)$ and $\mathcal{A}(6)$ is the empty set – see $\mathcal{T}_2$ and $\mathcal{T}_3$ in figure 8.

Algorithm 7 – Algorithm MPS

/* X — Set of candidates to the following shortest path. */

Compute $\mathcal{T}_t^\star$
Compute $\bar{c}_{ij}$, for any arc $(i,j) \in \mathcal{A}$
For the computed reduced costs, rearrange the set of arcs of $(\mathcal{N}, \mathcal{A})$ in the sorted forward star form
$p_1 \leftarrow$ shortest path from s to t /* p_1 is a path of $\mathcal{T}_t^\star$ */
$k \leftarrow 1$
$X \leftarrow \{p_k\}$
$\mathcal{T}_k \leftarrow \{p_k\}$
While $k < K$ **and** $X \neq \emptyset$
do begin
  $X \leftarrow X - \{p_k\}$
  $v_k \leftarrow$ deviation node of p_k
  for each node $v \in p^k_{v_k t}$
  do begin
    if $\mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v) \neq \emptyset$
    then begin
      $(v,x) \leftarrow$ first arc in the set $\mathcal{A}(v) - \mathcal{A}_{\mathcal{T}_k}(v)$
      $q \longleftarrow p^k_{sv} \diamond \{v,(v,x),x\} \diamond p^\star_{xt}$
      $X \longleftarrow X \cup \{q\}$
      $q_{vt} \longleftarrow \{v,(v,x),x\} \diamond p^\star_{xt}$
      $\mathcal{T}_k \longleftarrow \mathcal{T}_k \cup \{q_{vt}\}$ /* In such a way that $p^k_{sv} \diamond q_{vt}$ is a path of $\mathcal{T}_k$ */
    end
  end
  $k \leftarrow k+1$
  $p_k \leftarrow$ shortest path in X
end

The computational complexity of MPS algorithm is $\mathcal{O}(m \log m + Kn)$, after the determination of $\mathcal{T}_t^\star$ and when a worst case analysis is considered. In fact, computing reduced cost and rearranging the network in the sorted forward star form can be done in $\mathcal{O}(m\ logm)$; moreover, in the worst case no more than n different nodes has to be considered after the deviation node, when new candidate paths are being added to the set X. That is, for each k, n new candidate paths are generated which can be achieved in $\mathcal{O}(n)$.

5 Conclusion

The ranking shortest paths problem was studied in this paper and two classes of algorithms were presented. One class comprises all the algorithms supported by the Principle of Optimality and the other comprises algorithms that built the optimal solution using a set of candidates to the following path. Obviously, the efficiency of the algorithms in this last class depends on the number of candidates in the set; that is, algorithms will be more efficient if less paths are generated.

Eppstein's algorithm is that one who presents the best theoretical computational complexity, followed by MPS algorithm. However, in practice it appears that MS algorithm performs better, [16, 18], but MPS algorithm appears to be a good alternative. A more or less exaustive computational work about all the algorithms is needed to confirm, or not, this assumption. This will be the subject of a future paper.

One great advantage of algorithms in the last class is the possibility of being adapted for ranking loopless paths without loosing its practical efficiency, [15, 18]. This will be presented too in a future paper.

We must point out that all the algorithms presented (or refered) in the paper, can be used either for directed or undirected networks.

Acknowledgments:
Research developed within **CISUC** ("Centro de Informática e Sistemas da Universidade de Coimbra") and partly supported by the **PRAXIS XXI** Project of **JNICT** ("Junta Nacional de Investigação Científica e Tecnológica").

References

[1] J.A. Azevedo, M.E.O.S. Costa, J.J.E.R.S. Madeira, and E.Q.V. Martins. An algorithm for the ranking of shortest paths. *European Journal of Operational Research*, 69:97–106, 1993.

[2] J.A. Azevedo, J.J.E.R.S. Madeira, E.Q.V. Martins, and F.M.A. Pires. A shortest paths ranking algorithm, 1990. Proceedings of the Annual Conference AIRO'90, Models and Methods for Decision Support, Operational Research Society of Italy, 1001-1011.

[3] J.A. Azevedo, J.J.E.R.S. Madeira, E.Q.V. Martins, and F.M.A. Pires. A computational improvement for a shortest paths ranking algorithm. *European Journal of Operational Research*, 73:188–191, 1994.

[4] R.E. Bellman. On a routing problem. *Quarterly Applied Mathematics*, 1:425–447, 1958.

[5] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shorthest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–196, 1996.

[6] E.V. Denardo and B.L. Fox. Shortest route methods: reaching, pruning and buckects. *Operations Research*, 27:161–186, 1979.

[7] N. Deo and C. Pang. Shortest paths algorithms: taxonomy and annotation. *Networks*, 9:275–323, 1979.

[8] R. Dial, G. Glover, D. Karney, and D. Klingman. A computational analysis of alternative algorithms and labelling techniques for finding shortest path trees. *Networks*, 9:215–348, 1979.

[9] E. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:395–412, 1959.

[10] S.E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17:395–412, 1969.

[11] D. Eppstein. Finding the k shortest paths. SIAM Journal on Computing (to appear).

[12] G. Gallo and S. Pallotino. Shortest path methods: a unifying approach. *Mathematical Programming Study*, 26:38–64, 1986.

[13] R. Hoffman and R. R. Pavley. A method for the solution of the n th best path problem. *Journal of the Association for Computing Machinery*, 6:506–514, 1959.

[14] E.Q.V. Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18:123–130, 1984.

[15] E.Q.V. Martins, M.M.B. Pascoal, and J.L.E. Santos. A new algorithm for ranking loopless paths. Submitted, 1997.

[16] E.Q.V. Martins and J.L.E. Santos. A new shortest paths ranking algorithm. Submitted, 1996.

[17] E.F. Moore. The shortest path through a maze. Proceedings of the International Symposium on the Theory of Switching, (1959), Harvard University Press, 285–292.

[18] M.M.B. Pascoal. Algoritmos para a enumeração dos k trajectos mais curtos, 1997. (Dissertação de Mestrado; Departamento de Matemática; Universidade de Coimbra).

[19] D. Shier. Computational experience with an algorithm for finding the k shortest paths in a network. *Journal of Research of the NBS*, 78:139–164, 1974.

[20] D. Shier. Interactive methods for determining the k shortest paths in a network. *Networks*, 6:151–159, 1976.

[21] D. Shier. On algorithms for finding the k shortest paths in a network. *Networks*, 9:195–214, 1979.

[22] J.Y. Yen. Shortest path network problems. (Mathematical Systems in Economics, Heft 18), Hain, (1975), Meisennheim am Glan.

[23] J.Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.