4OR 1: 121–133 (2003)

DOI: 10.1007/s10288-002-0010-2

A new implementation of Yen's ranking loopless paths algorithm

Ernesto Q.V. Martins[2, ⋆], **Marta M.B. Pascoal**[1,2, ⋆⋆]

[1] Centro de Informática e Sistemas
[2] Departamento de Matemática, Universidade de Coimbra, Apartado 3008, 3001-454 Coimbra, Portugal (e-mail: marta@mat.uc.pt)

Received: December 2001 / Revised version: October 2002

Abstract. Yen's algorithm is a classical algorithm for ranking the K shortest loopless paths between a pair of nodes in a network. In this paper an implementation of Yen's algorithm is presented. Both the original algorithm and this implementation present $\mathcal{O}(Kn(m + n \log n))$ computational complexity order when considering a worst-case analysis. However, computational experiments are reported, which allow to conclude that in practice this new implementation outperforms two other, Perko's implementation and a straightforward one.

Keywords: network, path, loopless path, paths ranking.

AMS classification: 05C38, 05C85, 68R10, 90C27.

1 Introduction

The problem of determining the K shortest paths, or the ranking of the K shortest paths between a pair of nodes in a network is due to Hoffman and Pavley (1959), who proposed an algorithm for solving it. In this problem, for a given integer $K \geq 1$, is intended to determine successively the shortest path, the second shortest path, …, until the K-th shortest path between the given pair of nodes.

Since 1959 numerous articles on this subject have been published, several of them suggesting algorithms for solving the K shortest paths problem. We cite some

⋆ Sadly, the author passed away in November, 2000.

⋆⋆ The research of Marta Pascoal was developed within CISUC and partially supported by the Portuguese Ministry of Science and Technology (MCT), under PRAXIS XXI Project of JNICT.

of those articles (Dreyfus 1969; Eppstein 1998; Jiménez and Varó 1999; Martins 1984; Martins and Santos 2000; Martins et al. 1997, 2001; Shier 1976), and refer a very complete K shortest paths bibliography, available online at: http://liinwww.ira.uka.de/bibliography/Theory/k-path.html.

Solving this problem is important for several optimisation problems, and it is often used ranking paths by non-decreasing order of their costs, until one or more paths with small cost and satisfying some constraints are determined.

A related problem is the K shortest loopless paths problem, where it is intended to enumerate only paths without repeated nodes, i.e., loopless paths. The additional constraint imposed implies that this problem does not verify the Optimality Principle (Martins et al. 1999); therefore, although being similar to the general problem, ranking loopless paths is significantly harder. Yet, algorithms for enumerating K shortest loopless paths can be found in Carraresi and Sodini (1983); Katoh et al. (1982); Martins et al. (1997); Yen (1971). The first of these algorithms was presented in 1971 by Yen (Lawler 1972; Yen 1971, 1975), and comparative analysis of its practical performance with other approaches in the literature can be found in Hadjiconstantinou and Christofides (1999); Martins et al. (1997); Perko (2000).

In this work an implementation of Yen's ranking loopless paths algorithm is presented. While the usual implementation follows straightforwardly the classical version of this algorithm, the new one uses a different approach since the nodes of each k-th shortest loopless path are analysed by reverse order. Although both implementations have complexity $\mathcal{O}(Kn(m + n \log n))$, in a worst-case analysis, the new one revealed a better performance in the computational experiments made to estimate their behaviours in an average-case.

This paper is divided in five sections. In Sect. 2 some definitions, notation and the formulation of the K shortest loopless paths problem are introduced. In Sect. 3 a class of algorithms for ranking paths and loopless paths is described. In the following section Yen's algorithm is briefly described, being also presented its new implementation. Finally, in Sect. 5, computational experiments are reported, testing the behaviour of the variant presented against a straightforward implementation of Yen's algorithm and an implementation of the same algorithm due to Perko (1986).

2 Preliminaries

Let $(\mathcal{N}, \mathcal{A})$ denote a given network, where $\mathcal{N} = \{v_1, \ldots, v_n\}$ is a finite set, the elements of which are called nodes, and $\mathcal{A} = \{a_1, \ldots, a_m\} \subset \mathcal{N} \times \mathcal{N}$ is a finite set, the elements of which are called arcs, where each arc a_k can be represented by (v_i, v_j), with $v_i \neq v_j$. When (v_i, v_j) is an ordered (unordered) pair for any $(v_i, v_j) \in \mathcal{A}$, $(\mathcal{N}, \mathcal{A})$ is said to be a directed (undirected) network.

Let i and j be two nodes of $(\mathcal{N}, \mathcal{A})$. A path p from i to j in $(\mathcal{N}, \mathcal{A})$ is a sequence of the form $p = \langle v'_1, a'_1, v'_2, \ldots, a'_{\ell-1}, v'_\ell \rangle$, such that:

- $v'_k \in \mathcal{N}$ for any $k \in \{1, \ldots, \ell\}$;

- $a'_k = (v'_k, v'_{k+1}) \in \mathcal{A}$ for any $k \in \{1, \ldots, \ell - 1\}$;
- $i = v'_1$ and $j = v'_\ell$.

Nodes i and j are called the initial and terminal nodes of path p, respectively. To simplify the notation paths will be represented only by their nodes. Each single node is a degenerated path, having no arcs.

A cycle or loop in $(\mathcal{N}, \mathcal{A})$ is a path from one node to itself ($i = j$) where all nodes, except i and j, are different. Therefore a path is said to be loopless when it does not have repeated nodes.

$\mathcal{P}_{ij}$ will denote the set of paths in $(\mathcal{N}, \mathcal{A})$ from node i to node j. Given x and y, two nodes of $p \in \mathcal{P}_{ij}$, then $q \in \mathcal{P}_{xy}$ is said to be a sub-path of p if it coincides with p from x until y. Such a path will be denoted by $\text{sub}_p(x, y)$.

Each arc (i, j) is associated with a real value, called the arc cost, and denoted by c_{ij}. Let c be a function defined by

$$\begin{aligned} c \; : \bigcup_{i,j \in \mathcal{N}} \mathcal{P}_{ij} &\longrightarrow \mathbb{R} \\ p \quad &\longrightarrow c(p) = \sum_{(x,y) \in p} c_{xy} \end{aligned},$$

for any path p in $(\mathcal{N}, \mathcal{A})$; $c(p)$ is known as the path p cost.

The concatenation of two paths, $p \in \mathcal{P}_{ij}$ and $q \in \mathcal{P}_{j\ell}$, is denoted by $p \diamond q$ and it is the path from i to ℓ formed by path p followed by q.

Let s and t be two different nodes of $(\mathcal{N}, \mathcal{A})$ named, respectively, the initial and the terminal nodes in the network, and let $\mathcal{P}$ denote the set $\mathcal{P}_{st}$.

Given a positive integer K, in the K shortest loopless paths problem a set $\mathcal{P}_K = \{p_1, \ldots, p_K\} \subseteq \mathcal{P}$ has to be determined, such that:

- p_k is loopless, for any $k \in \{1, \ldots, K\}$;
- $c(p_k) \leq c(p_{k+1})$, for any $k \in \{1, \ldots, K - 1\}$;
- $c(p_K) \leq c(p)$, for any loopless path $p \in \mathcal{P} - \mathcal{P}_K$;
- p_k is determined before p_{k+1}, for any $k \in \{1, \ldots, K - 1\}$.

In the following it will be assumed, with no loss of generality, that $(\mathcal{N}, \mathcal{A})$ is a directed network and that $\mathcal{P}_{si} \neq \emptyset$, $\mathcal{P}_{it} \neq \emptyset$, for any $i \in \mathcal{N}$. It will also be assumed the existence of at least K loopless paths in $(\mathcal{N}, \mathcal{A})$ and that all cycles in the network have non-negative cost.

3 Deviation algorithms

Some of the algorithms known in the literature for ranking the K shortest paths between a pair of nodes are based on the construction of a "pseudo"-tree. In fact it is not a tree as it is usually defined since it can contain repeated nodes. However, it can be seen in such a way, since the same node is distinguished when belonging

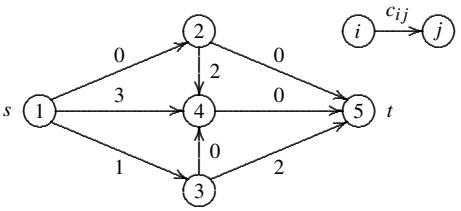

Fig. 1. Network $(\mathcal{N}, \mathcal{A})$

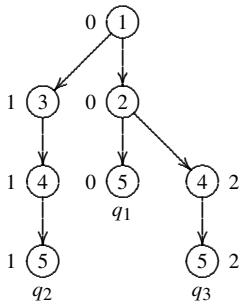

Fig. 2. Tree of paths

to different paths. This "pseudo"-tree (as well as the deviation node of a path) is defined recursively in the following.

Given $q_1, \ldots, q_k \in \mathcal{P}$, the deviation node of path $q_{k+1} \in \mathcal{P} - \{q_1, \ldots, q_k\}$, denoted by $d(q_{k+1})$, is the farthest node from the initial one (when concerning the number of intermediate nodes), among those where q_{k+1} deviates from each path $q_1, \ldots, q_k$. By definition $d(q_1) = s$.

A path $q_1 \in \mathcal{P}$ forms a tree with only one path. Assume $q_1, \ldots, q_k \in \mathcal{P}$ form a "pseudo"-tree of k paths, and let $q_{k+1} \in \mathcal{P} - \{q_1, \ldots, q_k\}$. Then, the "pseudo"-tree of $k+1$ paths $q_1, \ldots, q_{k+1}$ is defined by adding the branch $\text{sub}_{q_{k+1}}(d(q_{k+1}), t)$ to the "pseudo"-tree of k paths.

When a "pseudo"-tree is formed only by paths not containing cycles, it is said to be a loopless paths "pseudo"-tree, or simply a loopless paths tree.

Figure 2 represents the tree of paths $q_1 = \langle 1, 2, 5\rangle$, $q_2 = \langle 1, 3, 4, 5\rangle$ and $q_3 = \langle 1, 2, 4, 5\rangle$ from 1 to 5 in the network depicted in Fig. 1. The number close to each node in the tree of Fig. 2 represents the cost of the tree path from s until that node. It should be noticed that q_1, q_2 and q_3 do not contain repeated nodes, therefore they are loopless paths. Since these are the shortest loopless paths in that network, the tree in Fig. 2 is the tree of the three shortest loopless paths from 1 to 5 in $(\mathcal{N}, \mathcal{A})$.

According to the previous definition, the deviation node of q_1 is the initial node, that is $d(q_1) = 1$. Since q_2 separates from q_1 in node 1, $d(q_2) = 1$. Moreover q_3 coincides with q_1 from node 1 until node 2, loopless path $\langle 1, 2\rangle$, and it coincides

with q_2 only in the initial node, null loopless path $\langle 1 \rangle$. Therefore its deviation node is $d(q_3) = 2$.

One of the classes of algorithms for ranking shortest paths and shortest loopless paths is known as the class of deviation algorithms (Martins et al. 1997). In the case of the K shortest loopless paths problem, these algorithms construct the K shortest loopless paths tree. However, most of the times, this implies the determination of a "super"-tree of the K shortest loopless paths tree, i.e., a tree which contains the K shortest loopless paths, but which may also contain other paths.

In order to determine this tree, deviation algorithms use a set X of (loopless) paths, which are candidates to the next (loopless) path to be determined. The set of candidate paths is initialized with the shortest (loopless) path from s to t in $(\mathcal{N}, \mathcal{A})$, which may be determined by any shortest path algorithm. After that the element with lower cost is repeatedly picked up from X and new (loopless) paths are generated, and then stored in X. Throughout the algorithm the paths in X represent the several paths in the tree which is being formed. This procedure is repeated until K paths are determined, so that the (loopless) paths picked from X are $p_1, p_2, \ldots, p_K$. Whenever p_k is chosen, for any $k \in \{1, \ldots, K\}$, new (loopless) paths, candidates to p_j with $j > k$, are computed. In order to prevent the recalculation of paths only nodes in $\text{sub}_{p_k}(d(p_k), t)$ are analysed. Besides, when analysing $d(p_k)$ the arcs starting in $d(p_k)$ which belong to other (loopless) paths already determined should not be used.

4 Yen's algorithm

Yen's algorithm is a deviation algorithm where only loopless paths are determined. Considering that the k-th shortest loopless path is of the form $p_k = \langle s = v_1^k, v_2^k, \ldots, v_{\ell_k}^k = t \rangle$, for every node v_i^k to analyse, the shortest loopless path p, which deviates from p_k in v_i^k, is computed. Loopless path p_k is said to be the parent of p and node v_i^k is its deviation node. In order to calculate only loopless paths, nodes of $\text{sub}_{p_k}(s, v_{i-1}^k)$ should not be repeated. Therefore they are temporarily removed from $(\mathcal{N}, \mathcal{A})$ and the shortest loopless path from v_i^k to t in the resulting modified network is determined.

In a straightforward implementation of Yen's algorithm every node of p_k from $d(p_k)$ until $v_{\ell_k - 1}^k$ is analysed. For each of those nodes, for instance v_i^k, a specific shortest loopless path p has to be computed. This loopless path should have the form $p = \text{sub}_{p_k}(s, v_i^k) \diamond q$, where $q \in \mathcal{P}_{v_i^k t}$, the first arc of which is not (v_i^k, v_{i+1}^k), and which hasn't been computed yet. In order to calculate such path p, the network has to be modified by removing the nodes in $\text{sub}_{p_k}(s, v_{i-1}^k)$, the arc (v_i^k, v_{i+1}^k) and all arcs starting in v_i^k which have been deleted when p_k was computed. After these modifications, q is obtained by solving the shortest path problem from v_i^k to t in the resulting network. Finally the original network is restored.

Although some of the modifications can be maintained from one analysed node to the following one, each one implies the resolution of one new shortest path problem, which has $\mathcal{O}(m + n \log n)$ complexity when considering Dijkstra's algorithm (1959) (assuming that only non-negative arcs costs exist). Thus, considering the worst-case for this algorithm, when each $p_1, \ldots, p_K$ demands the analysis of n nodes, its theoretical complexity order is $\mathcal{O}(Kn(m + n \log n))$ (Lawler 1972).

In the following sub-section a new implementation of Yen's algorithm will be described, which differs from the previous one by the order of analysis of the nodes in p_k.

4.1 An implementation of Yen's algorithm

In the following the shortest tree rooted at $x \in \mathcal{N}$ will be denoted by $\mathcal{T}_x$ and the loopless path from $i \in \mathcal{N}$ to x in $\mathcal{T}_x$ will be denoted by $\mathcal{T}_x(i)$. When x is the terminal node, $\mathcal{T}_x$ represents the tree of the shortest paths from every node to t, otherwise it represents the tree of the shortest path from x to every node. The cost of path $\mathcal{T}_t(i)$ will be represented by π_i and will be called node i's label, for any $i \in \mathcal{N}$.

The natural order for analysing the nodes of each p_k seems to be from the deviation node $d(p_k)$ until t, following its order in the path. This implies several changes in the network and the resolution of shortest path problems (as many as the nodes in $\text{sub}_{p_k}(d(p_k), v^k_{\ell_k-1})$. Although both the straightforward implementation and the one which will now be described analyse exactly the same nodes for each p_k, in the last one the order of analysis is reversed. As it will be shown that allows us to solve only one shortest path problem, and to replace the remaining resolutions of shortest path problems by other operations, which, most of the times, are expected to be less complex.

Let $v^k_{\ell_k-1}$ be the first node to analyse in p_k; then a loopless path with the form $p = \text{sub}_{p_k}(s, v^k_{\ell_k-1}) \diamond q$ has to be computed, where q is the shortest loopless path from $v^k_{\ell_k-1}$ until t, without nodes in $\text{sub}_{p_k}(s, v^k_{\ell_k-2})$ and $(v^k_{\ell_k-1}, t)$. Thus, the nodes and arcs mentioned are deleted from $(\mathcal{N}, \mathcal{A})$, and (loopless) path q is determined. If a labeling algorithm is applied to obtain q, a shortest tree is computed and two options may be considered. One is to use the forward star form and construct the tree of the shortest paths from $v^k_{\ell_k-1}$ to every node in the network (then $q = \mathcal{T}_{v^k_{\ell_k-1}}(t)$); while the other is to use an adaptation of the usual shortest path algorithm and the reverse star form, and build the tree of the shortest paths from every node to t (and $q = \mathcal{T}_t(v^k_{\ell_k-1})$). Recalling that the next node to analyse is $v^k_{\ell_k-2}$, it can be noticed that, as in the straightforward implementation, the first option implies the complete construction of $\mathcal{T}_{v^k_{\ell_k-2}}$, which means that a new shortest path problem would have to be solved. On the other hand, with the second option, analysing $v^k_{\ell_k-2}$ consists in suitably changing $\mathcal{T}_t$, according to the reinsertion of that node.

Let $v^k_i \neq d(p_k)$ be a node to analyse in p_k, and let us assume that $\text{sub}_{p_k}(s, v^k_{i+1}) \diamond \mathcal{T}_t(v^k_{i+1})$ has been determined, where $\mathcal{T}_t$ refers to a network obtained from $(\mathcal{N}, \mathcal{A})$

by deleting the nodes in the loopless path $\text{sub}_{p_k}(s, v_i^k)$ and the arc (v_{i+1}^k, v_{i+2}^k). In order to analyse v_i^k, a similar tree has to be computed which is slightly different from the previous one, since it may contain both the arc (v_{i+1}^k, v_{i+2}^k) and the node v_i^k. This may force some nodes' labels in $\mathcal{T}_t$ to change.

Let us first consider the reinsertion of (v_{i+1}^k, v_{i+2}^k) in the network. This reinsertion implies correcting the tree $\mathcal{T}_t$ since this is the solution of a similar problem but subject to less restrictions, which means that the label of some nodes may be improved. Thus, after restoring (v_{i+1}^k, v_{i+2}^k), all nodes x in the modified network, such that there is at least one loopless path from x to v_{i+1}^k in that network, have to be analysed again.

Let us now consider reinserting node v_i^k in the network which is being used. Since this node has been deleted, all arcs in the network starting in v_i^k should be analysed. If such arcs exist, then the label of v_i^k can be calculated. This new label may allow other labels to be improved and therefore the process used when (v_{i+1}^k, v_{i+2}^k) was reinserted should be repeated, once again in order to analyse nodes x in the network, such that there is at least one loopless path from x to v_i^k. If, after these two phases, $\pi_{v_i^k}$ is finite, then $\mathcal{T}_t(v_i^k)$ is defined and the new loopless path generated as candidate to future shortest loopless path should be $p = \text{sub}_k(s, v_i^k) \diamond \mathcal{T}_t(v_i^k)$; otherwise i hasn't been labeled and no path p exists under the imposed restrictions.

It should be noticed that when $v_i^k = d(p_k)$ the same procedure should be used, but keeping in mind that besides $(d(p_k), v_{i+1}^k)$, also all the arcs in the network deleted when p_k was generated have to be, once again, deleted from the network.

Concerning the data structure used to represent the network $(\mathcal{N}, \mathcal{A})$, it should be noticed that given $i \in \mathcal{N}$, the arcs (i, j) and (j, i) of the network, with $j \in \mathcal{N}$, have to be analysed. Therefore, both the forward star form (for knowing all the arcs of the first type) and the backward star form (for the second type) should be used to represent $(\mathcal{N}, \mathcal{A})$, (Dial et al. 1979). On this concrete aspect the former implementation outperforms this one since it only demands representing the network in the forward star form.

Schematic descriptions of the proposed implementation and of the procedures used when labeling and correcting labels of the nodes in $\mathcal{T}_t$ are presented in the following Algorithm and in Procedures 1 and 2.

Algorithm – *New implementation of Yen's algorithm*

$p \longleftarrow$ shortest (loopless) path from s to t in $(\mathcal{N}, \mathcal{A})$; $d(p) \longleftarrow s$;
$X \longleftarrow \{p\}$; $k \longleftarrow 0$;
`While` $(X \neq \emptyset$ and $k < K)$ `Do`
  $k \longleftarrow k + 1$;
  $p_k \longleftarrow$ shortest loopless path in X;
  $X \longleftarrow X - \{p_k\}$;
  $\pi_i \longleftarrow +\infty$, for any $i \in \mathcal{N}$;
  Remove loopless path p_k, except node t, from the network;

Remove arcs $(d(p_k), i)$, $i \in \mathcal{N}$, of $p_1, \ldots, p_{k-1}$ from the network;
$\mathcal{T}_t \longleftarrow$ shortest tree rooted at t in the network;
For $(v_i^k \in \{v_{\ell_k-1}^k, \ldots, d(p_k)\})$ Do
Restore node v_i^k in the network;
Calculate $\pi_{v_i^k}$ using forward star form; /* Calculate label of v_i^k */
If ($\pi_{v_i^k}$ is defined) Then
Correct labels of v_i^k successors using backward star form;
$p \longleftarrow \text{sub}_{p_k}(s, v_i^k) \diamond \mathcal{T}_t(v_i^k)$; $d(p) \longleftarrow v_i^k$; $X \longleftarrow X \cup \{p\}$;
EndIf
Restore (v_i^k, v_{i+1}^k) in the network;
If $(\pi_{v_i^k} > \pi_{v_{i+1}^k} + c_{v_i^k v_{i+1}^k})$ Then
$\pi_{v_i^k} \longleftarrow \pi_{v_{i+1}^k} + c_{v_i^k v_{i+1}^k}$;
Correct labels of v_i^k successors using backward star form;
EndIf
EndFor
Restore nodes and arcs of path p_k from s to $d(p_k)$ in the network;
Restore arcs $(d(p_k), i)$, $i \in \mathcal{N}$, of $p_1, \ldots, p_{k-1}$ in the network;
EndWhile

Procedure 1 – *Calculate π_i using forward star form*

For $((i, j) \in \mathcal{A})$ Do
If $(\pi_i > \pi_j + c_{ij})$ Then $\pi_i \longleftarrow \pi_j + c_{ij}$;
EndFor

Procedure 2 – *Correct labels of x successors using backward star form*

$list \longleftarrow \{x\}$;
Repeat
$i \longleftarrow$ element of $list$; $list \longleftarrow list - \{i\}$;
For $((j, i) \in \mathcal{A})$ Do
If $(\pi_j > \pi_i + c_{ji})$ Then
$\pi_j \longleftarrow \pi_i + c_{ji}$;
$list \longleftarrow list \cup \{j\}$;
EndIf
EndFor
Until $(list = \emptyset)$;

Let us consider again the network represented in Fig. 1. Yen's algorithm begins by determining $p_1 = \langle 1, 2, 5 \rangle$. When this path is chosen in X, all nodes and arcs of p_1, except node $t = 5$, are removed from $(\mathcal{N}, \mathcal{A})$ and then the shortest tree rooted at 5 is computed, Fig. 3a.

After that, nodes $v_2^1 = 2$ and $d(p_1) = v_1^1 = 1$ are analysed. The first one to be analysed is $v_2^1 = 2$ and it is intended to know the shortest path from 2 to 5. It should be noticed that arc (2, 5) can not be used in the new path, so it is maintained deleted from $(\mathcal{N}, \mathcal{A})$. Then, using the forward star form, arcs starting in 2 are

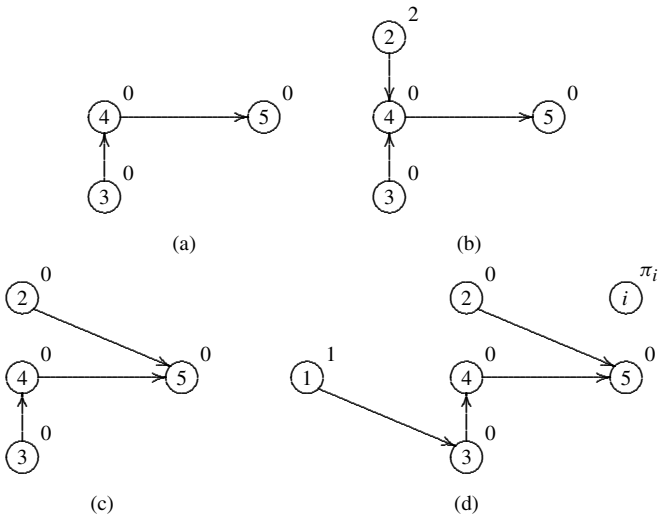

Fig. 3. Steps of the new implementation of Yen's algorithm

analysed and 2 is labeled from 4, with $\pi_2 = \pi_4 + c_{2,4}$. Using the backward star form the arcs incident in 2 are analysed and their labels are improved, if possible. The resulting shortest tree rooted at 5 is represented in Fig. 3(b). Since 2 has been labeled, the shortest loopless path which deviates from p_1 in 2, $\langle 1, 2, 4, 5\rangle$, is stored in X because it is candidate to p_j, for some $j \geq 2$.

The next node to be analysed is 1, but before that arc (2, 5) has to be restored in $(\mathcal{N}, \mathcal{A})$. Reinserting this arc implies correcting the label of node 2, and eventually others, using the backward star form. Figure 3(c) represents the shortest tree after this correction. Finally, node 1 is restored and analysed. The obtained tree, Fig. 3(d), allows to conclude that $\langle 1, 3, 4, 5\rangle$ is the shortest path deviating from p_1 in 1, so this path is also stored in X. In the following step $X = \{\langle 1, 2, 4, 5\rangle, \langle 1, 3, 4, 5\rangle\}$, and the shortest loopless path is picked from X. Therefore, $p_2 = \langle 1, 3, 4, 5\rangle$, and the algorithm continues until enough loopless paths (depending on K) are determined.

When using a straightforward implementation of Yen's algorithm, both paths $\langle 1, 3, 4, 5\rangle$ and $\langle 1, 2, 4, 5\rangle$ are still computed, but resulting from two independent resolutions of shortest path problems.

For storing the loopless paths generated throughout the algorithm, both implementations use a data structure based on the tree formed by those loopless paths, noticing that loopless path p can be characterized simply by its deviation node $d(p)$, the loopless path $\text{sub}_p(d(p), t)$ and its parent loopless path. It is not necessary to store the initial part of p, $\text{sub}_p(s, d(p))$, nor the entire path p, once they can be known recursively. More details about this structure can be found in Martins et al. (1999).

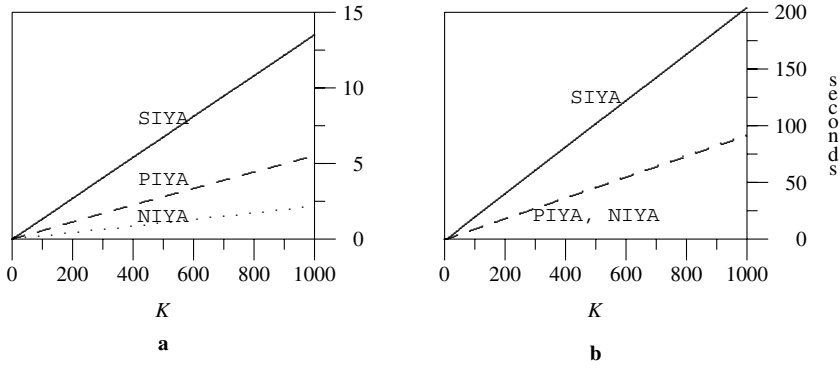

Fig. 4. Random Networks, 5000 nodes. **a** 7500 arcs. **b** 100000 arcs

In this implementation, the deletion of nodes and arcs of p_k from the network is replaced by its reinsertion. This allows to replace each resolution of a shortest loopless path problem by labeling and correcting labels of some nodes. In general, this number of nodes is expected to be smaller than the total number of nodes in the network when labels are corrected, therefore it is also expected to achieve better results using the new implementation. This conclusion seems to be confirmed by the computational tests we made, presented in the next section. However, in a worst-case scenario, all nodes are relabeled, thus in that case the number of operations performed by the new implementation is still $\mathcal{O}(Kn(m + n \log n))$ as for the straightforward one (considering, once again, that Dijkstra's algorithm is used).

5 Computational experiments

In this section experimental results are reported, where the practical performance of three implementations of Yen's ranking loopless paths algorithm is compared, namely: a straightforward implementation where nodes in p_k are analysed by the usual order (from $d(p_k)$ to t), `SIYA`, an implementation proposed by Perko (1986), `PIYA`, and the new implementation, presented in Sect. 4.1, `NIYA`.

All the implementations were coded in C language and the tests were carried out on a AMD Athlon 1.3 GHz computer with 512 megabytes of RAM, running over Suse Linux 7.3. The plots are result of the average CPU running times for solving 15 problems generated with the same parameters, except for a seed value. In `SIYA` and `NIYA` a label correcting algorithm was used, where the list of nodes to be scanned is manipulated as a FIFO for solving the shortest path problem and labeling additional nodes, while in `PIYA` a special representation of that list is used, avoiding several initializations (for further details consult Perko 1986).

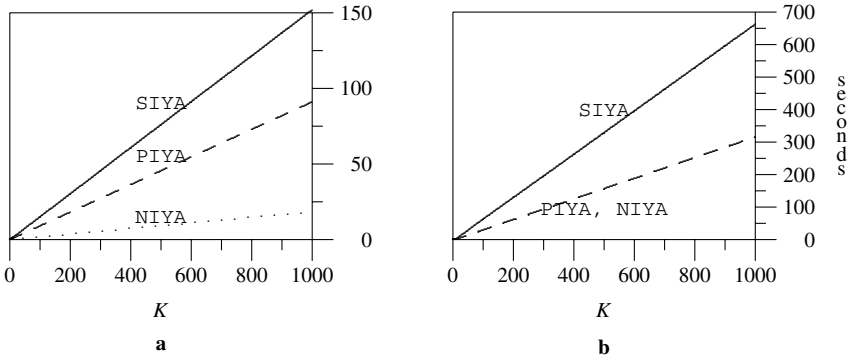

Fig. 5. Random networks, 10000 nodes. **a** 15000 arcs. **b** 200000 arcs

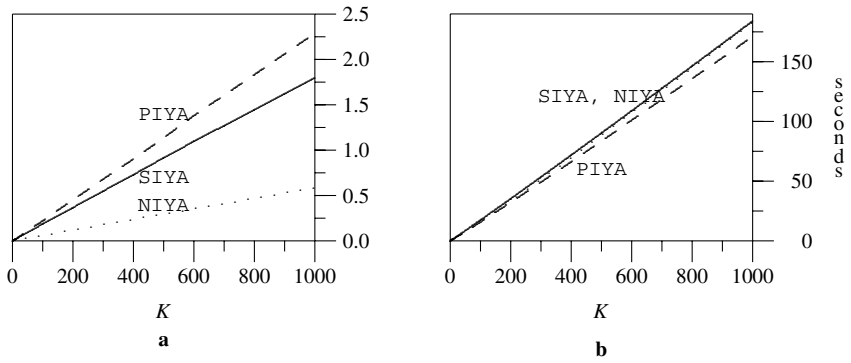

Fig. 6. Complete networks. **a** 100 nodes. **b** 500 nodes

In these tests, $K = 1000$ loopless paths were ranked in random networks with 5000 and 10000 nodes, and densities of 1.5, 2, 15 and 20, where the density or average degree of a network is $d = m/n$. Complete networks were also used, considering 100 and 500 nodes. The cost of each arc is randomly calculated and it is uniformly distributed in [0, 1000].

Tables 1 and 2 show the running times obtained by the three implementations, when considering random networks with 5000 and 10000 nodes, respectively, varying the density in {1.5, 2, 15, 20}, and computing $K = 10$ (Tables 1a and 2a) and $K = 1000$ (Tables 1b and 2b) loopless paths. These tables also present the ratios obtained when comparing the straightforward and Perko's implementation with the implementation of Yen's algorithm described in the last section. Based on those results one can see that the CPU times increase with d and with n. It can also be noticed that for small as well as for large values of K, the new implementation

Table 1. Running times (in seconds) for random networks, 5000 nodes and **a** $K = 10$, **b** $K = 1000$

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	1.3120	1.6700	15.5013	19.3973
PIYA	0.5507	0.6233	6.1093	8.6107
NIYA	0.2153	0.3600	6.0633	8.5600
$\frac{\text{SIYA}}{\text{NIYA}}$	6.0938	4.6389	2.5566	2.2660
$\frac{\text{PIYA}}{\text{NIYA}}$	2.5578	1.7313	1.0076	1.0059

a

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	13.5033	16.7647	159.7660	203.9327
PIYA	5.5173	6.4347	64.3440	90.9207
NIYA	2.1727	3.6467	63.8980	92.2740
$\frac{\text{SIYA}}{\text{NIYA}}$	6.2150	4.5972	2.5003	2.2101
$\frac{\text{PIYA}}{\text{NIYA}}$	2.5394	1.7645	1.0070	0.9853

b

Table 2. Running times (in seconds) for random networks, 10000 nodes and **a** $K = 10$, **b** $K = 1000$

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	15.0233	13.8620	51.3047	63.0060
PIYA	8.9753	4.7927	18.2660	29.7540
NIYA	1.8180	2.6820	20.6487	28.4273
$\frac{\text{SIYA}}{\text{NIYA}}$	8.2636	5.1685	2.4846	2.2164
$\frac{\text{PIYA}}{\text{NIYA}}$	4.9370	1.1787	0.8846	1.0467

a

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	151.8560	140.3507	540.8880	661.8706
PIYA	90.9720	48.7987	191.7907	314.9373
NIYA	18.3840	27.2840	225.0020	316.1820
$\frac{\text{SIYA}}{\text{NIYA}}$	8.2602	5.1441	2.4039	2.0933
$\frac{\text{PIYA}}{\text{NIYA}}$	4.9484	1.7885	0.8524	0.9961

b

outperforms the classical and the one suggested by Perko, being the classical the one with the worst running times. However, the difference in the average running times seems to be less evident when the density of the network is higher. In this last case (higher density values) the results presented by PIYA and NIYA are very similar, and PIYA even shows a better performance in some of the problems.

Figures 4 and 5 represent the CPU times variation with the number of ranked loopless paths, from $K = 1$ until $K = 1000$, in small ($d = 1.5$) and higher ($d = 20$) density random networks with $n \in \{5000, 10000\}$, while the times in Fig. 6 refer to complete networks with $n \in \{100, 500\}$. From these figures it is notorious the linear dependence of the running times on the value of K, for all the implementations. Once again one can see that the classical implementation presented the worst results, followed by Perko's and then the new implementation, although PIYA and NIYA had close running times for higher density values (as already seen in Tables 1 and 2). In complete networks the results are analogous, though depending on the number of nodes. Thus, NIYA has the best CPU times for networks with less nodes, and the times are similar for the three implementations when that number increases.

References

Carraresi P, Sodini C (1983) A binary enumeration tree to find k shortest paths. Proc. 7th Symposium on Operations Research, *Methods of Operations Research* 45:177–188

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

Dijkstra E (1959) A note on two problems in connection with graphs. *Numerical Mathematics* 1:395–412
Dreyfus SE (1969) An appraisal of some shortest-path algorithms. *Operations Research* 17: 395–412
Eppstein D (1998) Finding the *k* shortest paths. *SIAM Journal on Computing* 28:652–673
Hadjiconstantinou E, Christofides N (1999) An efficient implementation of an algorithm for finding *k* shortest simple paths. *Networks* 34(2):88–101
Hoffman R, Pavley RR (1959) A method for the solution of the *n*th best path problem. *Journal of the Association for Computing Machinery* 6:506–515
Jiménez VM, Varó AM (1999) Computing the *k* shortest paths: a new algorithm and an experimental comparison. Proc. 3rd Workshop Algorithm Engineering (terra.act.uji.es/REA/papers/wae99.ps.gz)
Katho N, Ibaraki T, Mine H (1982) An efficient algorithm for *k* shortest simple paths. *Networks* 12:411–427
Lawler EL (1972) A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science* 18:401–405
Martins EQV (1984) An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research* 18:123–130
Martins EQV, Santos JLE (2000) A new shortest paths ranking algorithm. *Investigação Operacional* 20(1):47–62 (www.mat.uc.pt/∼equm/cientifices/investigacao/Artigos/k.ps.gz)
Martins EQV, Pascoal MMB, Santos JLE (1997) A new algorithm for ranking loopless paths. *Research Report, CISUC*. (www.mat.uc.pt/∼marta/Publicacoes/mps.ps.gz)
Martins EQV, Pascoal MMB, Santos JLE (1999) Deviation algorithms for ranking shortest paths. *Intern J Foundations Comp Sci* 10(3):247–263
Martins EQV, Pascoal MMB, Santos JLE (2001) A new improvement for a *k* shortest paths algorithm. *Investigaçao Operacional* 21:47–60 (www.mat.uc.pt/∼marta/Publicacoes/ms_improved.ps.gz)
Perko A (1986) Implementation of algorithms for *k* shortest loopless paths. *Networks* 16:149–160
Shier D (1976) Interactive methods for determining the *k* shortest paths in a network. *Networks* 6:151–159
Yen JY (1971) Finding the *k* shortest loopless paths in a network. *Management Science* 17:712–716
Yen JY (1975) Shortes path network problems. *Mathematical Systems in Economics* 18