A NEW IMPLEMENTATION OF YEN'S RANKING LOOPLESS PATHS ALGORITHM[1]

Ernesto de Queirós Vieira Martins(2)
and
Marta Margarida Braz Pascoal(1,2)

{*eqvm, marta*}*@mat.uc.pt*
(1) Centro de Informática e Sistemas
(2) Departamento de Matemática
Universidade de Coimbra
Apartado 3008
3001-454 Coimbra
PORTUGAL
Phone: +351 239 791150, Fax: +351 239 832568

October 2000

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. Although with no consequences in what concerns the computational complexity order when considering a worst-case analysis, $\mathcal{O}(Kn(m + n\log n))$, in practice this new implementation outperforms a straightforward one, as the reported comparative computational experiments allow us to conclude.

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

---

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 was due to Hoffman and Pavley, [6], 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 1951 numerous articles on this subject have been published, several of them suggesting algorithms for solving the K shortest paths problem. We cite some of those articles, [3, 4, 7, 10, 13, 12, 14, 16], and refer a very complete K shortest paths bibliography, online at the address `http://liinwww.ira.uka.de/bibliography/Theory/k-path.html`.

[1]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.

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

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. Although being similar to the general problem, ranking loopless paths is significantly harder since the additional constraint imposed causes the non verification of the Optimality Principle, [11]. Yet algorithms for enumerating K shortest loopless paths can be found in [8, 12, 17]. One of these algorithms was presented in 1971 by Yen, [9, 17, 18]. Comparative analysis of its pactical performance can be found in [5, 15, 12].

In this paper 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 slightly 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 proved to have a better performance in the computational experiments made to estimate their behaviours in an average-case.

This paper is divided in five sections. In Section 2 some definitions, notation and the formulation of the K shortest loopless paths problem are introduced. In Section 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 the new implementation proposed. Finally, in Section 5, computational experiments comparing the behaviours of the classical and the new implementations of Yen's algorithm are reported.

2 Preliminaries

Let $(\mathcal{N}, \mathcal{A})$ denote a given network, where $\mathcal{N} = \{v_1, \ldots, v_n\}$ is a finite set whose elements are called nodes and $\mathcal{A} = \{a_1, \ldots, a_m\} \subset \mathcal{N} \times \mathcal{N}$ is also a finite set, whose elements 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. A null path is formed only by a single node.

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 a path $p \in \mathcal{P}_{ij}$, 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 $\mathrm{sub}_p(x, y)$.

A real value will be associated with any $(i,j) \in \mathcal{A}$, called the arc (i,j) cost, and denoted by c_{ij}. Let c be a function defined by

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

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 is intended to determine a set $\mathcal{P}_K = \{p_1, \ldots, p_K\} \subseteq \mathcal{P}$ of paths, 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 what follows 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 there are no cycles in the network with negative costs. The k-th shortest loopless path from s to t will be denoted by p_k and it will be considered to be of the form $p_k = \langle s = v_1^k, v_2^k, \ldots, v_{\ell_k}^k = t \rangle$.

3 Deviation algorithms

Some of the known algorithms for ranking the K shortest paths between a pair of nodes in a given network 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, as long as the same node is distinguished when belonging to different paths. This "pseudo"-tree will be defined by construction when proving Lemma 3.1.

Lemma 3.1 *Given k, a positive integer, k paths between a pair of nodes in a network form a "pseudo"-tree.*

Proof: Let us consider a path q_1 from s to t in $(\mathcal{N},\mathcal{A})$; obviously q_1 forms a tree with only one path. Let us now assume that $q_1, \ldots, q_k$, k different paths between s and t in $(\mathcal{N},\mathcal{A})$, form a "pseudo"-tree of paths. Considering another path q_{k+1} from s to t in $(\mathcal{N},\mathcal{A})$, is easy to notice that q_{k+1} coincides with each $q_1, \ldots, q_k$ from the initial node s, until a certain node (which may be s), where it deviates from each one of those paths. From these nodes, the farthest from the initial one (when concerning the number of intermediate nodes) is denoted by $d(q_{k+1})$. This is the node where q_{k+1} deviates from all the previous paths $q_1, \ldots, q_k$. Since by assumption $q_1, \ldots, q_k$ form a "pseudo"-tree, then $q_1, \ldots, q_k, q_{k+1}$ have the same structure, forming a "pseudo"-tree with $k+1$ paths. $\square$

Given a path p, the node $d(p)$ defined in Lemma 3.1 is said to be the deviation node of p. By assumption $d(p_1) = s$.

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.

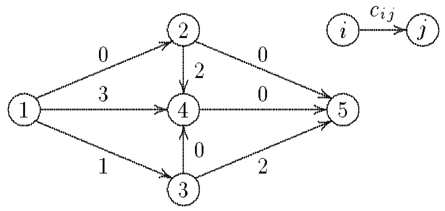

Figure 1: Network $(\mathcal{N}, \mathcal{A})$

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

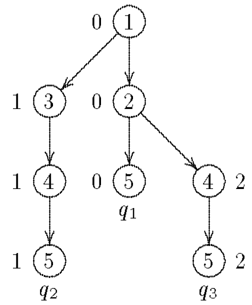

Figure 2: Tree of paths

According to what was previously said, 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, [12]. In the case of the K shortest loopless paths problem these algorithms intend to 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, candidates to the next (loopless) path to be determined. The set of candidate paths is initialized with p_1, the shortest (loopless) path from s to t in $(\mathcal{N}, \mathcal{A})$ (determined, for instance, by the algorithm proposed in [2]). In a general step the element with lower cost is picked out in X, by analysing it in a specific way, new (loopless) paths are generated, and X is updated. When the algorithm is used the paths in X represent the several paths in the tree which is being constructed. This procedure is repeated until p_k is determined. Thus the (loopless) paths successively picked out in X are $p_1, p_2, \ldots, p_K$. For any $k \in \{1, \ldots, K\}$, whenever p_k is chosen new (loopless) paths, candidates to p_j with $j > k$, are computed. In order to prevent the recalculation of paths only nodes in $\mathrm{sub}_{p_k}(d(p_k), t)$ 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 can be determined. Then, 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 compute only loopless paths, nodes of $\mathrm{sub}_{p_k}(s, v_{i-1}^k)$ 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 one of those nodes, for instance v_i^k, a specific shortest loopless path $p \in \mathcal{P}$ has to be computed. This loopless path should have the form $p = \mathrm{sub}_{p_k}(s, v_i^k) \diamond q$, with $q \in \mathcal{P}_{v_i^k t}$ whose first arc is not (v_i^k, v_{i+1}^k), and which hasn't been computed yet. In order to calculate p satisfying these conditions the given network has to be modified by deletion of all nodes in $\mathrm{sub}_{p_k}(s, v_{i-1}^k)$, 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 mantained from one analysed node to the following one, still each one implies the resolution of one shortest path problem, which has $\mathcal{O}(m + n \log n)$ complexity when considering Dijkstra's algorithm, [2]. 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 $\mathcal{O}(Kn(m + n \log n))$, [9].

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

4.1 An implementation of Yen's algorithm

As it was previously said, usually the nodes of each p_k are analysed from its deviation node $d(p_k)$, until t, following its order in the path. This implies several changes in the network and solving shortest paths problems (as many as the nodes in $\mathrm{sub}_{p_k}(d(p_k), t)$). Although the implementation which will now be described and the straightforward one both analyse the same nodes, in the former this analysis is made by reverse order, since reversing the order of analysing the nodes in each p_k allows to solve only one shortest path problem. The other resolutions of the shortest path problem are replaced by different operations, most of the times not so complex.

Let $v_{\ell_k-1}^k$ be the first node to analyse in p_k. Given $v_{\ell_k-1}^k$, a loopless path with the form $p = \mathrm{sub}_{p_k}(s, v_{\ell_k-1}^k) \diamond q$ has to be computed, where q is the shortest loopless path from $v_{\ell_k-1}^k$ until t without nodes in $\mathrm{sub}_{p_k}(s, v_{\ell_k-2}^k)$ and without arc $(v_{\ell_k-1}^k, t)$. Thus, all the nodes and arcs mentioned are deleted from $(\mathcal{N}, \mathcal{A})$, and after that (loopless) path q is determined. Using an adaptation of a shortest path algorithm, the tree of the shortest loopless paths from i to t, for any $i \in \mathcal{N}$, that is, the shortest tree rooted at t, can be computed and therefore that loopless path q can be obtained.

In what follows 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 $\mathcal{T}_x(i)$. The cost of path $\mathcal{T}_t(i)$ will be represented by π_i and will be called node i's label, for any $i \in \mathcal{N}$.

Obviously, another option would be to construct $\mathcal{T}_{v^k_{\ell_k-1}}$, and then notice that $q = \mathcal{T}_{v^k_{\ell_k-1}}(t)$. However this choice would imply once again the complete reconstruction of the tree of the shortest loopless paths when analysing the next node, and which is intended to avoid.

Let us now consider any given node to analyse in p_k, v^k_i, such that $v^k_i \neq d(p_k)$ 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 $\text{sub}_{p_k}(s, v_i)$ and the arc (v^k_{i+1}, v^k_{i+2}). In order to analyse node v^k_i a similar tree has to be computed which is slightly different from the previous one, since it may contain both the arc (v^k_{i+1}, v^k_{i+2}) and the node v^k_i. This may force some nodes' labels in $\mathcal{T}_t$ to change.

Let us first consider the re-insertion of arc (v^k_{i+1}, v^k_{i+2}) in the network. This re-insertion 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^k_{i+1}, v^k_{i+2}), all nodes x in the modified network such that there is at least one loopless path from x to v^k_{i+1} in that network have to be analysed again.

Let us now consider re-inserting node v^k_i in the network which is being used. Since this node has been deleted all arcs in the network and starting in v^k_i should be analysed. If such arcs exist then the label of v^k_i can be calculated. This new label may allow other labels to be improved and therefore the same process used when (v^k_{i+1}, v^k_{i+2}) was re-inserted 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^k_i. If after these two phases $\pi_{v^k_i}$ is finite, then $\mathcal{T}_t(v^k_i)$ is defined and the new loopless path generated should be $p_k = \text{sub}_k(s, v^k_i) \diamond \mathcal{T}_t(v^k_i)$; otherwise it is not possible to generate p under the imposed restrictions.

It should be noticed that when $v^k_i = d(p_k)$ the same procedure should be used but recalling that more than $(d(p_k), v^k_{i+1})$, also all the arcs in the network deleted when p_k was generated have to be, once again, deleted from the network.

In what concerns the data structure used to represent the network $(\mathcal{N}, \mathcal{A})$ it should be noticed that for any given $i \in \mathcal{N}$, the arcs of the network with the form (i, j), as well as the arcs of the network with the form (j, i), for any $j \in \mathcal{N}$, have to be analysed. Therefore, both forward star form (for knowing all the arcs of the first kind) and reverse star form (for the second kind) should be used to represent $(\mathcal{N}, \mathcal{A})$, [1].

It should be noticed that 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 Algorithm 1 and in Procedures 1.1 and 1.2.

Algorithm 1 – *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 $k \leq K)$ `Do`
  $k \longleftarrow k + 1$;
  $p_k \longleftarrow$ shortest loopless path in X;
  $X \longleftarrow X - \{p_k\}$;

```
    Remove loopless path sub_{p_k}(s, v^k_{ℓ_k−1}) from the network;
    Remove arcs (d(p_k), i), i ∈ N, of p_1, ..., p_{k−1} from the network;
    T_t ⟵ shortest tree rooted at t in the network;
    For (v^k_i ∈ {v^k_{ℓ_k−2}, ..., d(p_k)}) Do
      Restore node v^k_i in the network;
      Calculate π_{v^k_i} using forward star form;    /*  Calculate label of v^k_i  */
      If (π_{v^k_i} is defined) Then
        Correct labels of v^k_i successors using reverse star form;
        p ⟵ sub_{p_k}(s, v^k_i) ◇ T_t(v^k_i); d(p) ⟵ v^k_i; X ⟵ X ∪ {p};
      EndIf
      Restore (v^k_i, v^k_{i+1}) in the network;
      If (π^k_{v_i} > π_{v^k_{i+1}} + c_{v^k_i v^k_{i+1}}) Then
        π^k_{v_i} ⟵ π_{v^k_{i+1}} + c_{v^k_i v^k_{i+1}};
        Correct labels of v^k_i successors using reverse 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 ∈ N, of p_1, ..., p_{k−1} in the network;
  EndWhile
```

Procedure 1.1 – *Calculate* π_i *using forward star form*

```
  For ((i, j) ∈ A) Do
    If (π_i > π_j + c_ij) Then π_i ⟵ π_j + c_ij;
  EndFor
```

Procedure 1.2 – *Correct labels of* x *successors using reverse star form*

```
  list ⟵ {x};
  Repeat
    i ⟵ element of list; list ⟵ list − {i};
    For ((j, i) ∈ A) Do
      If (π_j > π_i + c_ji) Then
        π_j ⟵ π_i + c_ji;
        If (j ∉ list) Then list ⟵ list ∪ {j};
      EndIf
    EndFor
  Until (list = ∅);
```

Let us consider again the network represented in Figure 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 the terminal node, $t = 5$, are removed from $(\mathcal{N}, \mathcal{A})$ and then the shortest tree rooted at $t = 5$ is computed, Figure 3(a).

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 mantained removed from $(\mathcal{N}, \mathcal{A})$. Then, using the forward star form the arcs beginning in 2 are analysed and 2 is labeled from 4, with $\pi_2 = \pi_4 + c_{2,4}$. Using

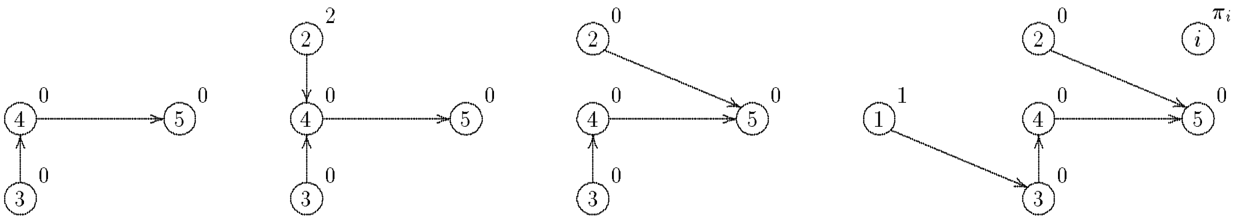

Figure 3: Steps of the new implementation of Yen's algorithm

the reverse star form the arcs inciding in 2 are analysed and their labels are improved, if possible. The resulting shortest tree rooted at 5 is represented in Figure 3(b). Since 2 is labeled, the shortest loopless path which deviates from p_1 in 2 is $\langle 1,2,4,5\rangle$ and it is stored in X since it is candidate to p_j, for some $j \geq 2$.

The next node to be analysed is node 1 but before that arc $(2,5)$ has to be restored in $(\mathcal{N},\mathcal{A})$. Re-inserting this arc implies correcting the label of node 2 and possibly other ones using the reverse star form. After this correction the shortest tree is the one in Figure 3(c). Finally node 1 is restored and analysed. The obtained tree, Figure 3(d), allows to conclude that $\langle 1,3,4,5\rangle$ is the shortest path which deviates from p_1 in 1. Once again this path is 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 chosen in X. Therefore $p_2 = \langle 1,3,4,5\rangle$ and the algorithm continues until enough loopless paths are determined.

When using a straightforward implementation 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 paths problems.

Concerning the data structure used to store the loopless paths successively determined, both implementations use the same one, which is based on the tree structure formed by those loopless paths. More details about this structure can be found in [11]. So, loopless path p can be characterized only by knowing its deviation node $d(p)$, the loopless path $\mathrm{sub}_p(d(p),t)$ and its parent loopless path. Thus, recursively, it can also be known the loopless path $\mathrm{sub}_p(s,d(p))$ and therefore all loopless path p.

In this implementation the deletion of nodes and arcs of p_k is replaced by its re-insertion in the network. 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 that the total number of nodes in the network when labels are corrected, thus it is also expected to achieve better results using the new implementation, rather than the former one. This conclusion appears to be confirmed by the computational tests made, presented in the next section. However, when considering the worst-case all nodes are relabeled, thus in this case the complexity of the operations performed by this new implementation will also be $\mathcal{O}(Kn(m+n\log n))$ as for the straightforward one.

5 Computational Experiments

In this section experimental results are reported, comparing the practical performance of two different implementations of Yen's ranking loopless paths algorithm: the straightforward implementation, SIYA, and the new one, NIYA, presented in Sub-Section 3.1.

Both implementations were coded in C and the tests were carried out on a Celeron 533 MHz computer with 192 megabytes of RAM running over Suse Linux 6.1. The curves in the graphics are result of the CPU running times for solving 15 problems generated with the same parameter, except for a seed value.

The tests presented used 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].

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	0.9320	1.1467	4.1813	5.0553
NIYA	0.1500	0.2000	1.1753	1.5600
$\frac{\texttt{SIYA}}{\texttt{NIYA}}$	6.2133	5.7334	3.5576	3.2406

(a)

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	106.4253	131.7627	545.4474	631.4187
NIYA	16.5927	23.8040	153.3913	211.0493
$\frac{\texttt{SIYA}}{\texttt{NIYA}}$	6.4140	5.5353	3.5559	2.9918

(b)

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

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	4.9373	4.1913	14.1247	16.9667
NIYA	0.4520	0.6100	3.0373	3.8727
$\frac{\texttt{SIYA}}{\texttt{NIYA}}$	10.9233	6.8710	4.6504	4.3811

(a)

	$d = 1.5$	$d = 2$	$d = 15$	$d = 20$
SIYA	476.9093	463.9240	670.4300	557.1820
NIYA	50.3860	67.6000	402.8460	497.5700
$\frac{\texttt{SIYA}}{\texttt{NIYA}}$	9.4651	6.8628	1.6642	1.1981

(b)

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

It is notorious the linear dependence of the running times obtained on the value of K, for both implementations. However it can be noticed for small as for large values of K that the new implementation suggested outperforms the classical one. The difference in the running times seems to be more evident when the density of the network is higher or, in the case of complete networks, when the number of nodes increases.

6 Conclusion

References

[1] 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.

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

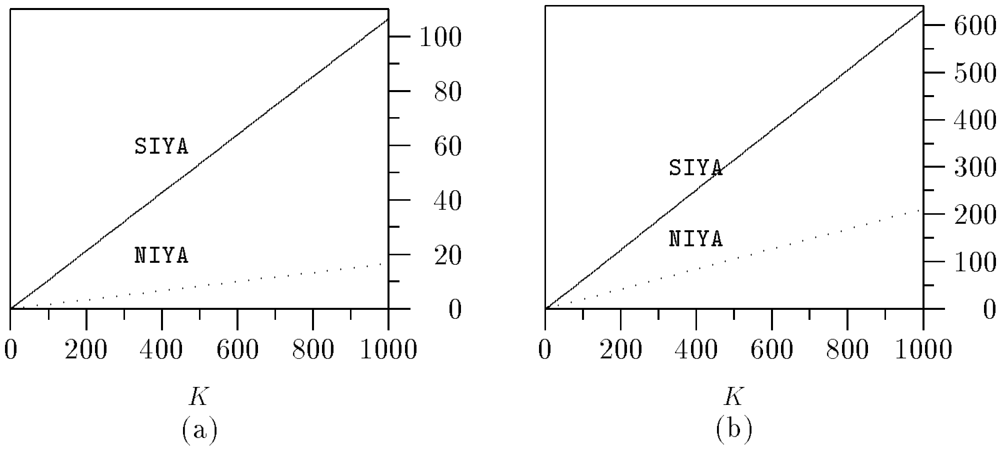

Figure 4: Random Networks, 5000 nodes and (a) 7500 arcs (b) 100000 arcs

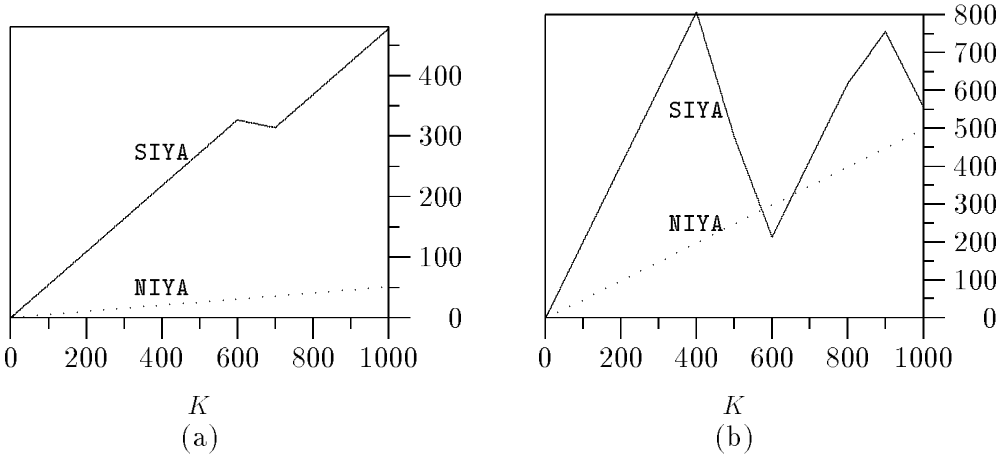

Figure 5: Random networks, 10000 nodes and (a) 15000 arcs (b) 200000 arcs

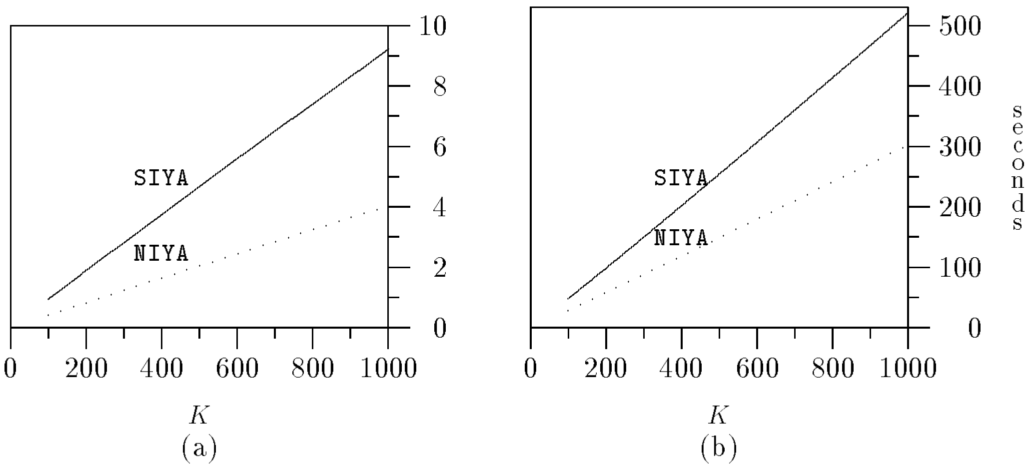

Figure 6: Complete networks, (a) 100 nodes and (b) 500 nodes

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

[4] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28:652–673, 1998.

[5] E. Hadjiconstantinou and N. Christofides. An efficient implementation of an algorithm for finding k shortest simple paths. *Networks*, 34(2):88–101, 1999.

[6] 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.

[7] V. M. Jiménez and A. M. Varó. Computing the k shortest paths: a new algorithm and an experimental comparison. In *Proc. 3rd Workshop Algorithm Engineering*, July 1999. (`http://terra.act.uji.es/REA/papers/wae99.ps.gz`)

[8] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12:411–427, 1982.

[9] E. L. Lawler. 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, 1972.

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

[11] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. Deviation algorithms for ranking shortest paths. *The International Journal of Foundations of Computer Science*, 10:(3):247–263, 1999. (`http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/deviation.ps.gz`)

[12] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. An algorithm for ranking loopless paths. *CISUC*, Technical Report 99/007, 1999. (`http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/MPS_novo.ps.gz`)

[13] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. A new improvement for a k shortest paths algorithm. Submitted to *Investigação Operacional*, 1999. (`http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/IMP_MS.ps.gz`)

[14] E. Q. V. Martins and J. L. E. Santos. A new shortest paths ranking algorithm. *Investigação Operacional*, 20:(1):47–62, 2000. (`http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/K.ps.gz`)

[15] A. Perko. Implementation of algorihms for k shortest loopless paths. *Networks*, 16:149–160, 1986.

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

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

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