A NEW IMPROVEMENT FOR A K SHORTEST PATHS ALGORITHM

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

{eqvm,marta,zeluis}@mat.uc.pt
Centro de Informática e Sistemas
Departamento de Matemática
Universidade de Coimbra
PORTUGAL

November 1999

Abstract:

The K shortest paths problem is a well known network optimization problem where it is intended to rank the K shortest paths between an initial and a terminal node in a network. The first algorithm for solving this problem appeared by the fifties and since then several other algorithms have been proposed. These algorithms can be divided into two classes: one based on the Optimality Principle and another based on the determination of a tree of shortest paths. Moreover, in the first of these classes there can be considered labeling algorithms and deletion path algorithms.

In this paper an improvement for a known deletion path algorithm is presented which results in the improvement of its execution time complexity when the worst case analysis is considered. Comparative computational experiments are also reported, allowing possible conclusions about the obtained performances when the average case is considered.

Keywords: path, ranking, deletion path algorithms

1 Introduction

The K shortest paths problem is a well known network optimization problem where it is intended to rank the K shortest paths between an initial and a terminal node in a network. The first algorithm for solving this problem was proposed by Hoffman and Pavley in 1959, [9], and ever since other papers presenting algorithms have been proposed among which we refer [6, 7, 10, 11, 13, 16, 17]. A very complete bibliography on the problem, due to Eppstein, can be found in the URL address `http://liinwww.ira.uka.de/bibliography/Theory/k-path.html`.

These algorithms can be divided into two classes: one based on the Optimality Principle generalization and another based on the determination of a tree formed by shortest paths, [14, 15]. Moreover, among the algorithms based on the Optimality Principle there can be considered labeling algorithms and deletion path algorithms (also known as algorithms of alternatives).

In this paper an improvement for a known deletion path algorithm is presented resulting in a decrease of its execution time complexity when the worst case analysis is considered. Comparative computational experiments are also reported, allowing possible conclusions about the obtained performances when the average case is considered.

The reader is assumed to be familiarized with some basic network optimization concepts and, mainly, with the deletion path algorithm known as MS algorithm, [16]. Nevertheless, in Sections 2 and 3 some of the notation and definitions used in this paper are introduced and the problem is described. Section 4 contains a brief overview on deletion path algorithms being the improvement of MS algorithm, [16], described in the following one. Finally, in Section 6 the computational complexity of MS algorithm improvement is studied and some comparative numerical results with its previous version are presented.

2 Notation and definitions

Let $(\mathcal{N}, \mathcal{A})$ denote a given network, where $\mathcal{N} = \{v_1, \ldots, v_n\}$ is a finite set with n elements called nodes and $\mathcal{A} = \{a_1, \ldots, a_m\} \subseteq \mathcal{N} \times \mathcal{N}$ is a finite set with m elements called arcs. Sometimes it will simply be used i to represent node v_i. Each arc $a_k \in \mathcal{A}$ can also be identified by a pair (i, j), where $i, j \in \mathcal{N}$.

Let i and j be two nodes of $(\mathcal{N}, \mathcal{A})$; if all pairs (i, j) in $(\mathcal{N}, \mathcal{A})$ are ordered, i.e., if all the arcs of the network are directed, $(\mathcal{N}, \mathcal{A})$ is said to be a directed network; if all the pairs (i, j) are not ordered, then $(\mathcal{N}, \mathcal{A})$ is said to be an undirected network. Without loss of generality, in what follows it will be considered that $(\mathcal{N}, \mathcal{A})$ is a directed network, since each undirected arc can be replaced by two arcs with opposite direction. It will also be assumed that there is at most a single arc between each pair of nodes of the network and that there are no arcs of the form (i, i), where $i \in \mathcal{N}$.

Given $i, j \in \mathcal{N}$, a path from node i to node j in $(\mathcal{N}, \mathcal{A})$ is an alternating sequence of nodes and arcs, of the form $p = \langle i = v'_1, a'_1, v'_2, \ldots, a'_{\ell-1}, v'_\ell = j \rangle$, where:

- $v'_k \in \mathcal{N}$, for every $k \in \{1, \ldots, \ell\}$;
- $a'_k \equiv (v'_k, v'_{k+1}) \in \mathcal{A}$, for every $k \in \{1, \ldots, \ell - 1\}$.

In order to simplify the notation a path will be represented only by its nodes; that is, $p = \langle v'_1, v'_2, \ldots, v'_\ell \rangle$ and for convenience, sometimes a single node is viewed as a path, the null path. The set of paths defined from i to j in $(\mathcal{N}, \mathcal{A})$ will be denoted by $\mathcal{P}_{ij}$. A loopless path from i to j is a path from i to j where all the nodes are different and a loop (or cycle) is a path from some node to itself, where all nodes are different except the first one which is also the last.

Given x and y, two nodes of a path p, the subpath of p from x to y, denoted by $\mathrm{sub}_p(x, y)$, is simply the path that coincides with p between these two nodes.

Let $p \in \mathcal{P}_{ix}$ be a path from i to x and $q \in \mathcal{P}_{xj}$ a path from x to j. The concatenation of p and q, denoted by $p \diamond q$, is a path from i to j formed by p until its node x and followed by path q from x to j. Given a cycle $\mathcal{C}$ it can be defined the k-cycle concatenation as $\mathcal{C}^k = \mathcal{C}^{k-1} \diamond \mathcal{C}$, with $k \geq 1$ and $\mathcal{C}^0$ a single node of cycle $\mathcal{C}$.

Let c_{ij} be a real value associated with arc (i, j) of $(\mathcal{N}, \mathcal{A})$, known as the (i, j) arc cost, and let $c(p) = \sum_{(i,j) \in p} c_{ij}$ (or, simply, $c(p) = \sum_p c_{ij}$), for a given path p in $(\mathcal{N}, \mathcal{A})$, be the p path cost.

Let s and t be two different nodes of $(\mathcal{N}, \mathcal{A})$, called initial and terminal node, respectively. In order to simplify, $\mathcal{P}$ will denote the set $\mathcal{P}_{st}$.

In what follows, with no loss of generality, it will be assumed that $\mathcal{P}_{si} \neq \emptyset$ and $\mathcal{P}_{it} \neq \emptyset$ holds, for every node i in $(\mathcal{N}, \mathcal{A})$. It will also be assumed that there are no arcs of the form (x, s) and (t, x), where $x \in \mathcal{N}$. This is assumed with no loss of generality, since a new initial node S, a new terminal node T, and the zero cost arcs (S, s) and (t, T) can be added to the network. These new nodes are usually known as super-source and super-sink and there can be easily defined a bijection between $\mathcal{P}_{st}$ and $\mathcal{P}_{ST}$. As a consequence, nodes s and t can be repeated in the path definition and each path has at least three arcs which is important in the proof of some of the theorems presented in what follows.

A path is said to be finite if it is a finite sequence of nodes (and arcs).

3 The K shortest paths problem

3.1 The problem

In the classical shortest path problem it is intended to determine a path p^* from s to t in $(\mathcal{N}, \mathcal{A})$ with minimum cost, that is, it is intended to determine $p^* \in \mathcal{P}$ such that $c(p^*) \leq c(p)$, for any $p \in \mathcal{P}$. Given an integer $K > 1$, the K shortest paths problem can be considered a generalization of the previous one where, beyond the determination of the shortest path in $\mathcal{P}$ it also have to be determined the second shortest path in $\mathcal{P}$, ..., until the K-th shortest path in $\mathcal{P}$. That is, denoting by p_i, $i \in \mathcal{N}$, the i-th shortest path from s to t in $(\mathcal{N}, \mathcal{A})$ it is intended to determine a set of paths, $\mathcal{P}_K = \{p_1, \dots, p_K\} \subseteq \mathcal{P}$, such that:

1. p_i is determined before p_{i+1}, for any $i = 1, \dots, K-1$;
2. $c(p_i) \leq c(p_{i+1})$, for any $i = 1, \dots, K-1$;
3. $c(p_K) \leq c(p)$, for any $p \in \mathcal{P} - \mathcal{P}_K$.

3.2 Finiteness, boundness and the Optimality Principle

As it is well known, when studying optimal problems, two very important concepts such as finiteness and boundness, have to be studied too, [12]. That is, conditions that assure the existence of a finite and bounded optimal solution have to be established, since only in this case it is possible to compute its optimal solution with an algorithm. Note that finiteness and boundness have to be studied jointly since optimal path problems exist where a non finite optimal solution is bounded, [12].

Considering the K shortest paths problem as a generalization of the shortest path problem these concepts should also be generalizations of the known ones for the simple problem and that will now be introduced. Since these notions and the introduced results are already known, no proofs will be presented. They can be found at [12, 15].

An instance of the shortest path problem is said to be finite if and only if there is a finite shortest path and it is said to be bounded if and only if its cost is finite. Under certain assumptions finiteness and boundness of the shortest path problem can be assured. This is stated in Theorem 1.

Theorem 1 *A shortest path problem is:*

1. *finite if and only if no cycle in the considered network has negative cost.*
2. *bounded if and only if no cycle in the considered network has negative cost.*

The labeling algorithm for any optimal path problem in general, which may assume several forms, is supported by the Optimality Principle, [12]. According to this principle: *Every subpath $sub_{p^*}(s, y)$ of p^*, the shortest path from s to t in $(\mathcal{N}, \mathcal{A})$, is a shortest path from s to y in $(\mathcal{N}, \mathcal{A})$.*

Under the conditions of Theorem 2 it can be proved, (see [12]), that this principle is verified.

Theorem 2 *A shortest path problem satisfies the Optimality Principle if and only if there are no cycles with negative cost in the given network.*

The concepts introduced above are naturally generalized to the K shortest paths problem. A K shortest paths problem is said to be finite if and only if there exists a finite K-th shortest path and it is said to be bounded if and only if this path has a finite cost.

The following results are analogous to Theorems 1 and 2 but concerning the K shortest paths problem.

Theorem 3 *A K shortest paths problem is:*

1. *finite if and only if no cycle in the given network has negative cost.*

2. *bounded if and only if no cycle in the given network has negative cost.*

The Optimality Principle for the general problem states that: *Every subpath $sub_{p_k}(s, y)$ of p_k, the k-th shortest path from s to t in $(\mathcal{N}, \mathcal{A})$ is a j_y-th shortest path from s to y in $(\mathcal{N}, \mathcal{A})$, for some $1 \leq j_y \leq k$.*

Analogously to the shortest path problem, conditions are established in Theorem 4 in order that the Optimality Principle is verified.

Theorem 4 *A K shortest paths problem satisfies the Optimality Principle if and only if there are no cycles with negative cost in the considered network.*

In order to assure finiteness and boundness of the problem, as well as the Optimality Principle being verified, from now on it will be assumed that $c(\mathcal{C}) = \sum_{\mathcal{C}} c_{ij} \geq 0$ for any cycle $\mathcal{C}$ in the network; that is, all the cycles in the considered network have non negative cost.

4 Deletion path algorithms

The first of a sequence of algorithms for ranking shortest paths was proposed by one of the authors in 1984, [11]. This algorithm is based on the successive deletion of paths from a network and it is supported by the Optimality Principle.

This first algorithm can be roughly described considering that the second shortest path from s to t in $(\mathcal{N}, \mathcal{A})$ is the shortest path from s to t in the network, as long as path p_1 no longer exists in the initial one. Repeating this procedure for $k > 1$, the k-th shortest path from s to t in $(\mathcal{N}, \mathcal{A})$ is the shortest path from s to t in the network, as long as none of the $k-1$ shortest paths, that is, $p_1, \ldots, p_{k-1}$, are allowed to exist in the network. So, denoting by $(\mathcal{N}_1, \mathcal{A}_1)$ the given network and by $(\mathcal{N}_k, \mathcal{A}_k)$ the network which is obtained from $(\mathcal{N}, \mathcal{A})$ in such a way that the set of paths from s to t is $\mathcal{P} - \{p_1, \ldots, p_{k-1}\}$, this algorithm can be sketched as presented bellow:

Algorithm 1 – *Deletion path algorithm*
$(\mathcal{N}_1, \mathcal{A}_1) \longleftarrow (\mathcal{N}, \mathcal{A})$;
$p_1 \longleftarrow$ shortest path from s to t in $(\mathcal{N}_1, \mathcal{A}_1)$
`For` $k \in \{2, \ldots, K\}$ `Do`
  $(\mathcal{N}_k, \mathcal{A}_k) \longleftarrow$ *Delete* p_{k-1} *from* $(\mathcal{N}_{k-1}, \mathcal{A}_{k-1})$;
  $p'_k \longleftarrow$ shortest path from s to t in $(\mathcal{N}_k, \mathcal{A}_k)$;
  $p_k \longleftarrow$ path in $(\mathcal{N}, \mathcal{A})$ corresponding to p'_k
`EndFor`

It should be noticed that an easy correspondence may have to be established in order that some path p' in $(\mathcal{N}_k, \mathcal{A}_k)$ can be identified with a path p in $(\mathcal{N}, \mathcal{A})$.

The key to this algorithm is the deletion of a path from a network. Shortly it can be said that this deletion is obtained by creating alternative nodes (that is, copies of these nodes) of the path to be removed. However, only some of the arcs related with these nodes are copied, in order that it is not possible to follow that path in the new network. The new computed paths, that is, the determined shortest paths, for instance p', correspond to a path p in the initial network, $(\mathcal{N}, \mathcal{A})$. Path p' is said to be an alternative for p and it may happen that these two paths are confounded. The procedure for deleting a path in a network is presented in Procedure 1.1.

Procedure 1.1 – *Delete* $p = \langle s = v_1, v_2, \ldots, v_\ell = t\rangle$ *from* $(\mathcal{N}, \mathcal{A})$ – *Martins' algorithm*

$(\mathcal{N}', \mathcal{A}') \longleftarrow (\mathcal{N}, \mathcal{A})$;
$\mathcal{N}' \longleftarrow \mathcal{N}' \cup \{v_2'\}$;
`For` $(i, v_2) \in \mathcal{A}$ such that $i \neq s$ `Do`
  $\mathcal{A}' \longleftarrow \mathcal{A}' \cup \{(i, v_2')\}$; $c_{iv_2'} \longleftarrow c_{iv_2}$
`EndFor`
`For` $j \in \{3, \ldots, \ell\}$ `Do`
  $\mathcal{N}' \longleftarrow \mathcal{N}' \cup \{v_j'\}$;
  $\mathcal{A}' \longleftarrow \mathcal{A}' \cup \{(v_{j-1}', v_j')\}$; $c_{v_{j-1}'v_j'} \longleftarrow c_{v_{j-1}v_j}$;
  `For` $(i, v_j) \in \mathcal{A}$ such that $i \neq v_{j-1}$ `Do`
    $\mathcal{A}' \longleftarrow \mathcal{A}' \cup \{(i, v_j')\}$; $c_{iv_j'} \longleftarrow c_{iv_j}$
  `EndFor`
`EndFor`

Recall that it was previously assumed, with no loss of generality, that $\ell \geq 3$. As it has already been mentioned, this is, in short, the former version of deletion path algorithms. However it is not very efficient, neither in the necessary memory space (since several similar copies of the same nodes and of the same arcs have to be done) nor in the execution time, since K shortest path problems have to be solved in networks whose number of nodes and number of arcs is growing up from one network to the following one.

The last improvement to this algorithm, known as MS algorithm, will now be described. In this last version – MS algorithm – the number of arcs in the network is constant and therefore the needed memory space decreases. In practice, it happens the same with the execution time.

The main MS algorithm idea is still the copy of the successive p_k paths nodes, for a later computation of paths p_j, with $j > k$. In what follows it will be tried to enumerate briefly the several improvements to Martins' algorithm. For more complete details the reader is addressed to [1, 2, 3, 16].

As presented in Algorithm 1, for each one of the determined p_k paths, each of its nodes is analysed and copied. The copy of node $i \in \mathcal{N}_k$ corresponds to a j-th shortest path from s to i, for some $1 \leq j \leq k$. This copy consists in a new node, somehow related with the corresponding node in the original network, and new arcs, each one associated to an arc which incides in the node corresponding to i in the original network.

It can be easily proved that, when determining the shortest path from s to t in the networks generated by the algorithm, each node label is permanent. So, in each step of the algorithm, labeling alternative nodes for every k-th shortest path can be made, instead of solving shortest paths problems. The label of node $i \in \mathcal{N}_k$, for some $k \geq 1$, that is, the cost of the shortest path from the initial node until node i will be denoted by π_i.

It should be pointed out that given some node v_i of a path p_k, if v_i precedes v_α, the first node of p_k where more than one arc incides, then there is no alternative path from s to v_i for $\text{sub}_{p_k}(s, v_i)$. Obviously, there is an alternative for v_α and for every node following v_α until t. Therefore it is enough to analyse the nodes of p_k following the first one in which more than one arc incides and to which no alternative has been determined yet. Thus the number of copies of nodes is reduced and the algorithm execution time is also decreased, since less labels have to be computed (and recomputed). Let v_β be the first of these nodes for which an alternative has not been determined yet. If v_β coincides with v_α, then it will be the first node to analyse; otherwise it will be analysed as an intermediate node.

The last improvement, the so called MS algorithm, appears by remarking that the set of the arcs inciding in the new alternative nodes is similar to the same set for the corresponding nodes in the previous network whose label is permanent and it is used only once in the determination of the label of its "copy" node. Thus, the replication of these arcs can be avoided since it is enough to associate the information referring the original node to each new node.

The MS algorithm is presented in Procedure 1.2.

Procedure 1.2 – *Delete* $p = \langle s = v_1, v_2, \ldots, v_\ell = t\rangle$ *from* $(\mathcal{N}, \mathcal{A})$ – *MS algorithm*

$v_\alpha \longleftarrow$ first node of p in which incides more than one arc;
`If` (v_α is defined) `Then`
  $v_\beta \longleftarrow$ first node in p from v_α for which no alternative has been determined yet;
  $(\mathcal{N}', \mathcal{A}') \longleftarrow (\mathcal{N}, \mathcal{A})$;
  $\gamma \longleftarrow \beta$;
  `If` ($v_\alpha = v_\beta$) `Then`
    $\mathcal{N}' \longleftarrow \mathcal{N}' \cup \{v'_\beta\}$;
    $\mathcal{A}' \longleftarrow \mathcal{A}' - \{(v_{\beta-1}, v_\beta)\}$;
    $\pi_{v'_\beta} \longleftarrow \min\{\pi_i + c_{iv_\beta} : (i, v_\beta) \in \mathcal{A}'\}$;
    $\gamma \longleftarrow \beta + 1$
  `EndIf`
  `For` $j \in \{\gamma, \ldots, \ell\}$ `Do`
    $\mathcal{N}' \longleftarrow \mathcal{N}' \cup \{v'_j\}$;
    $\mathcal{A}' \longleftarrow \mathcal{A}' - \{(v_{j-1}, v_j)\} \cup \{(v'_{j-1}, v_j)\}$; $c_{v'_{j-1}v_j} \longleftarrow c_{v_{j-1}v_j}$;
    $\pi_{v'_j} \longleftarrow \min\{\pi_i + c_{iv_j} : (i, v_j) \in \mathcal{A}'\}$
  `EndFor`
`Else`
  All paths from s to t have been determined
`EndIf`

5 Improved algorithm

In this section an improvement of MS algorithm, described in Algorithm 1 and in Procedure 1.2, is presented. In this improvement a particular order of the set of arcs inciding in each node of the network is used. This order decreases the number of the executed arithmetic operations, improving not only the theoretical computational complexity of the algorithm but also its practical efficiency.

Recalling MS algorithm, whenever a path p_k is determined, the nodes following a well determined node x have to be analysed, that is, some nodes have to be "copied" and "labeled". So, for such a node v_i, another node v'_i, has to be created, being $\pi_{v'_i} = \min\{\pi_j + c_{jv_i} : (j, v_i) \in \mathcal{A}_k\}$ its label. However, this labeling procedure demands an exhaustive search over the set of arcs whose head node is v_i; that is, all the arcs inciding in v_i have to be analysed. The technique that will be presented allows that each labeling is an immediate operation, improving the complexity of the previous algorithm.

From MS algorithm it can be concluded that given a node $v_i \in p_k$ to be copied, its alternative v'_i is labeled from another node u such that $(u, v_i) \in \mathcal{A}_k$ and $\pi_u + c_{uv_i} \leq \pi_v + c_{vv_i}$, for any $(v, v_i) \in \mathcal{A}_k$. Consequently, for any $k \geq 1$, let us consider $\mathcal{A}_k$ sorted as follows. For any $(i_1, j_1), (i_2, j_2) \in \mathcal{A}_k$:

$$\left\{\begin{array}{l} j_1 < j_2 \\ \quad or \\ j_1 = j_2 \quad and \quad \left\{\begin{array}{c} \pi_{i_1} + c_{i_1 j_1} < \pi_{i_1} + c_{i_1 j_1} \\ or \\ \pi_{i_1} + c_{i_1 j_1} = \pi_{i_1} + c_{i_1 j_1} \quad and \quad (i_1, j_1) \in \mathcal{T}_s \end{array}\right. \end{array}\right. \Longrightarrow (i_1, j_1) \text{ “} < \text{”} (i_2, j_2). \qquad (1)$$

For arcs $(i_1, j), (i_2, j) \in \mathcal{A}_k$, inciding in the same node, such that $\pi_{i_1} + c_{i_1 j} = \pi_{i_2} + c_{i_2 j}$ and $(i_1, j), (i_2, j) \notin \mathcal{T}_s$ the used order is not important. It can then be stated that under the conditions of relation (1), v'_i is labeled from a node u such that (u, v_i) follows the arc (v_{i-1}, v_i), preceding v_i

in path p_k in the ordered set $\mathcal{A}_k$; that is, u is the tail node of the arc inciding in v_i following (v_{i-1}, v_i) in set $\mathcal{A}_k$ ordered according to (1). Therefore, the set of arcs is ordered by non decreasing order of the head node of each arc, as in the reverse star form, [4], and each set of arcs inciding in the same node will be ordered by non decreasing order of the "label" that they can generate; that is, the set of arcs (i, j) inciding in node $j \in \mathcal{N}$, will be sorted according to the respective value of $\pi_i + c_{ij}$. In what follows this sort will be denominated the dynamic sorted reverse star form. A similar sort was used in the algorithm of Jiménez and Varó, [10].

Nevertheless, some changes have to be taken into account, when obtaining the sequence of networks $\{(\mathcal{N}_i, \mathcal{A}_i)\}_{i=1}^k$, since they can affect the set of arcs sort. It is known that creating node v_i', the copy of v_i, can imply deleting the arc preceding v_i in the analysed path (v_{i-1}, v_i), which does not change the order in the network; however, for some of the created alternatives, (v_{i-1}, v_i) is replaced by (v_{i-1}', v_i), which has the same cost of (v_{i-1}, v_i). Since the label of v_{i-1}' was computed in the previous step and in general, $\pi_{v_{i-1}'} > \pi_{v_{i-1}}$; therefore $\pi_{v_{i-1}'} + c_{v_{i-1}' v_i} > \pi_{v_{i-1}} + c_{v_{i-1} v_i}$. Thus, when replacing arc (v_{i-1}, v_i), it is necessary to insert (v_{i-1}', v_i) in the set of arcs whose head node is v_i, in such a way that the dynamic sorted reverse star form is maintained, that is, in order that relation (1) is satisfied.

This improvement is presented in Algorithm 2 and in Procedure 2.1.

Algorithm 2 – *Improved algorithm*
$(\mathcal{N}_1, \mathcal{A}_1) \longleftarrow (\mathcal{N}, \mathcal{A})$;
$\mathcal{T}_s \longleftarrow$ tree of the shortest paths from s to i, for any $i \in \mathcal{N}$;
$p_1 \longleftarrow$ path from s to t in $\mathcal{T}_s$; /* shortest path from s to t in $(\mathcal{N}_1, \mathcal{A}_1)$ */
Define $(\mathcal{N}_1, \mathcal{A}_1)$ in the dynamic sorted reverse star form;
`For` $k \in \{2, \ldots, K\}$ `Do`
  $(\mathcal{N}_k, \mathcal{A}_k) \longleftarrow$ *Delete* p_{k-1} *from* $(\mathcal{N}_{k-1}, \mathcal{A}_{k-1})$;
  $p_k' \longleftarrow$ shortest path from s to t in $(\mathcal{N}_k, \mathcal{A}_k)$;
  $p_k \longleftarrow$ path in $(\mathcal{N}, \mathcal{A})$ corresponding to p_k'
`EndFor`

In what follows, MS algorithm and its improvement will be briefly exemplified using the small network represented in Figure 1(a), where the initial node is $s = 1$ and the terminal node is $t = 4$. Note that neither the super initial nor the super terminal nodes have been included in this network since no arc incides in $s = 1$ and no arc emerges from $t = 4$.

Both algorithms begin by determining the tree of the shortest paths from s to every node in $\mathcal{N}$, Figure 1(b), which implies determining $p_1 = \langle 1, 2, 4 \rangle$. After that p_1 is deleted from $(\mathcal{N}, \mathcal{A}) = (\mathcal{N}_1, \mathcal{A}_1)$, thus determining $(\mathcal{N}_2, \mathcal{A}_2)$, represented in Figure 2, and consequently path $p_2 = \langle 1, 3, 2, 4 \rangle$. Note that, the computed path is, in fact, $\langle 1, 3, 2', 4' \rangle$, which in the original network corresponds to the given path, p_2. In order to determine this path, two new nodes have to be created, $2'$ and $4'$.

For labeling node $2'$ with MS algorithm

$$\pi_{2'} = \min\{\pi_i + c_{i2} : (i, 2) \in \mathcal{A}_2\} = \min\{\pi_i + c_{i2} : (i, 2) \in \{(3, 2)\}\} = 2$$

has to be computed; similarly, for labeling node $4'$,

$$\pi_{4'} = \min\{\pi_i + c_{i4} : (i, 4) \in \mathcal{A}_2\} = \min\{\pi_i + c_{i4} : (i, 4) \in \{(2', 4), (3, 4)\}\} = 2$$

has to be computed.

Now, when running MS algorithm improvement, after computing the tree in Figure 1(b) set $\mathcal{A}$ is sorted – see Table 1. Thus, when creating $2'$ arc $(1, 2)$ is removed and for labeling $2'$ arc $(3, 2)$ is chosen since it is the following one which incides in 2 (and also the unique in this case) and $\pi_{2'} = \pi_3 + c_{32} = 2$. Concerning the creation of $4'$, arc $(2, 4)$ is removed and it is replaced by $(2', 4)$. To maintain the order

Procedure 2.1 – *Delete* $p = \langle s = v_1, v_2, \ldots, v_\ell = t\rangle$ *from* $(\mathcal{N}, \mathcal{A})$ – *Improved algorithm*

```
v_α ⟵ first node in p where more than one arc incides;
If (v_α is defined) Then
   v_β ⟵ first node of p from v_α for which no alternative has been determined yet;
   (N', A') ⟵ (N, A);
   γ ⟵ β;
   If (v_α = v_β) Then
      N' ⟵ N' ∪ {v'_β};
      A' ⟵ A' − {(v_{β−1}, v_β)};
      (i, v_β) ⟵ first arc in A' inciding in v_β;
      π_{v'_β} ⟵ π_i + c_{iv_β};
      γ ⟵ β + 1
   EndIf
   For j ∈ {γ, ..., ℓ} Do
      N' ⟵ N' ∪ {v'_j};
      A' ⟵ A' − {(v_{j−1}, v_j)}; c_{v'_{j−1}v_j} ⟵ c_{v_{j−1}v_j};
      Insert (v'_{j−1}, v_j) in A' in order, satisfying (1);
      (i, v_j) ⟵ first arc in A' inciding in v_j;
      π_{v'_j} ⟵ π_i + c_{iv_j}
   EndFor
Else
   All paths from s to t have been determined
EndIf
```

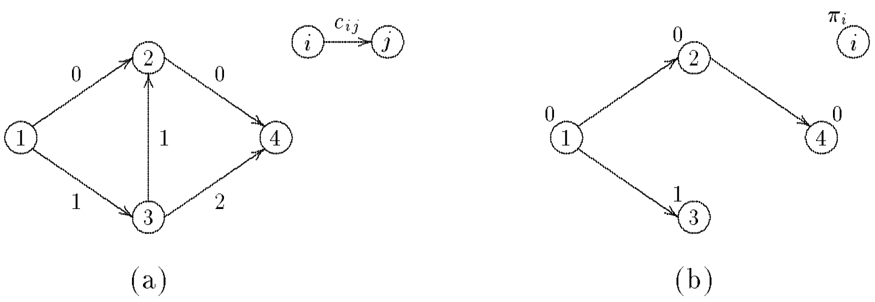

Figure 1: (a) Network $(\mathcal{N}, \mathcal{A})$, (b) respective shortest tree rooted at $s = 1$

(i,j)	(1,2)	(3,2)	(1,3)	(2,4)	(3,4)
$\pi_i + c_{ij}$	0	2	1	0	3

Table 1: The set $\mathcal{A}_1 = \mathcal{A}$ sorted according to relation (1)

(i,j)	(3,2)	(1,3)	(2′,4)	(3,4)
$\pi_i + c_{ij}$	2	1	2	3

Table 2: The set $\mathcal{A}_2$ sorted according to relation (1)

in $\mathcal{A}_2$, $(2',4)$ must be inserted in the changed set according to relation (1). Since $\pi_{2'} + c_{2'4} = 2$ and $\pi_3 + c_{34} = 3$, $(2',4)$ is inserted immediately before $(3,4)$. Then, when labeling node $4'$ arc $(2',4)$ is chosen, since it is the first arc inciding in node 4, and $\pi_{4'} = \pi_{2'} + c_{2'4} = 2$.

So, the second network to be constructed using both algorithms is $(\mathcal{N}_2, \mathcal{A}_2)$, in Figure 2, where ξ_i denotes the node from which $i \in \mathcal{N}$ has been labeled. However the set of arcs $\mathcal{A}_2$, obtained when using MS improvement, is sorted in a special way – represented in Table 2.

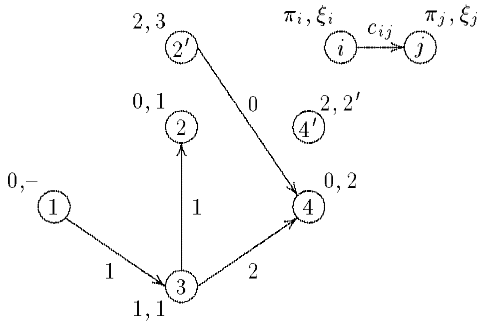

Figure 2: Network $(\mathcal{N}_2, \mathcal{A}_2)$

6 Computational experiments

It will now be studied the behavior of the improved algorithm beginning by looking over its computational complexity. However rather than the worst case we are interested in studying the performance for the average case, which leads us to present some comparative numerical results obtained by running implementations of MS algorithm and its improvement.

6.1 Computational complexity

In MS algorithm improvement, computing the minimum value of $\pi_i + c_{ij}$ for any arc (i,j) of the network which incide in node j is replaced by inserting, in order, an arc in that set, whenever a node is labeled. This operation seldom demands an exhaustive search which happens always with MS algorithm. The computational complexity of this improved algorithm can still be decreased using a suitable data structure for ordered insertions, such as binary heaps. It will now be presented a brief study of the computational complexity for both algorithms when considering the worst case analysis.

The worst case for these two algorithms happens when the shortest path which, in the absence of negative cycles, can be considered a loopless path from s to t in $(\mathcal{N}, \mathcal{A})$ contains every node of the network, its cost is zero and besides, there is an arc from t to s in the network, also with zero cost. In this case the k-th shortest path in $\mathcal{P}$ has cost zero and it can be expressed in the form $p_k = p_1 \diamond \mathcal{C}^{k-1}$, for any $k \geq 2$, where $\mathcal{C} = \langle t, s \rangle \diamond p_1$, that is $p_k = p_1 \diamond \overbrace{\langle t, s \rangle \diamond p_1 \diamond \cdots \diamond \langle t, s \rangle \diamond p_1}^{k-1 \text{ times}}$. It should be noticed that, even if the existence of cycles $\mathcal{C}$ such that $c(\mathcal{C}) = 0$ is not allowed, in order to avoid solutions of the form $q_1 \diamond \mathcal{C}^k \diamond q_2$, still an example of a k-th shortest path with the form given above can still be found, (see [14]).

Both algorithms begin by computing the shortest path in the network, which takes $\mathcal{O}(m + n \log n)$ time using heaps and Dijkstras' algorithm, [8, 5], considering only non-negative arcs costs. After that, in each step it is necessary to create alternatives for the nodes of p_k. For MS algorithm the creation of these alternatives requires the analysis of all the arcs inciding in the corresponding node of the initial network. So, in the worst case and for this algorithm, all the arcs in the network have to be analysed and therefore its complexity, in the worst case, is $\mathcal{O}(Km + n \log n)$.

For MS algorithm improvement described in this paper, after determining the shortest path, the set of arcs is sorted. For sorting that set it becomes necessary to sort the set of m_i arcs which incide in every node i, which takes $\mathcal{O}(m_i \log m_i)$ using binary heaps. Therefore the execution time for this sorting is $\sum_{i=1}^{n} m_i \log m_i \leq \sum_{i=1}^{n} m_i \log n = m \log n$. After that, it is necessary to sort the set of arcs inciding in every node of each path p_k, which in the case of complete newtworks, which is the worst, takes $\mathcal{O}(\log n)$. It should be noticed that after these sorting every node labeling is immediate. So the running time for this algorithm is $\mathcal{O}(m + Kn \log n)$, or simply $\mathcal{O}(Kn \log n)$, since usually $Kn > m$.

In what concerns the memory space, if p_k has also the given form, in MS algorithm the set of arcs always has at most m elements, and the set of nodes has the original ones and all those created alternatives; that is, the set of nodes has at most Kn elements. So, both MS algorithm and its improvement need $\mathcal{O}(m + Kn)$ memory space.

6.2 Execution times

As it was said before, besides the worst case it is interesting to analyse the behaviour of the presented algorithms for an average case. From the previous paragraph one would believe that, considering an average case, MS algorithm performance would be much better than its computational complexity. Even so, MS algorithm improvement seems to outperform its previous version, the MS algorithm.

In what follows the implementations, in C language, for the described algorithms are compared. The tests that will be presented ran on a Pentium II, over Linux, with a 266 MHz processor and 64 MB of RAM.

The presented execution times are average execution times for the resolution of 15 problems for each kind of network. Random networks with 10000 nodes and considering several densities were generated, where the density of a network is the ratio $d = m/n$. Density values were small (1.5 and 2, in Figure 3 and Figure 4) and higher (15 and 20, in Figure 5 and Figure 6).

It should be noticed that MS improvement seems to outperform MS algorithm, specially for high density networks. In fact that is the case in which, for each alternative node created, MS exhaustive search time becomes significant, versus the linear inserting time needed for its improvement.

We must point out the apparent linearity with K of the execution times, although MS improvement presents a lower slope.

The implemented code for MS improvement used linear insertion; so, it is expected that the execution times may decrease when using binary heaps. This is intended to be a future work, describing more complete computational experiments, including codes for different algorithms.

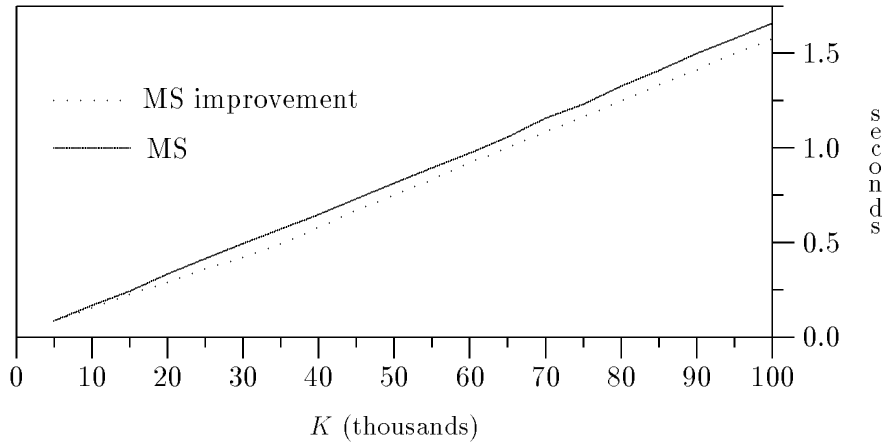

Figure 3: Random Networks, 10000 nodes and 15000 arcs

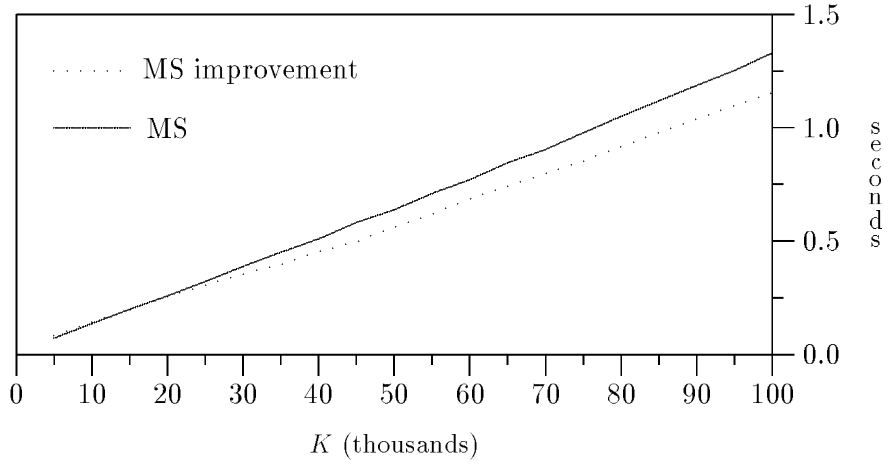

Figure 4: Random Networks, 10000 nodes and 20000 arcs

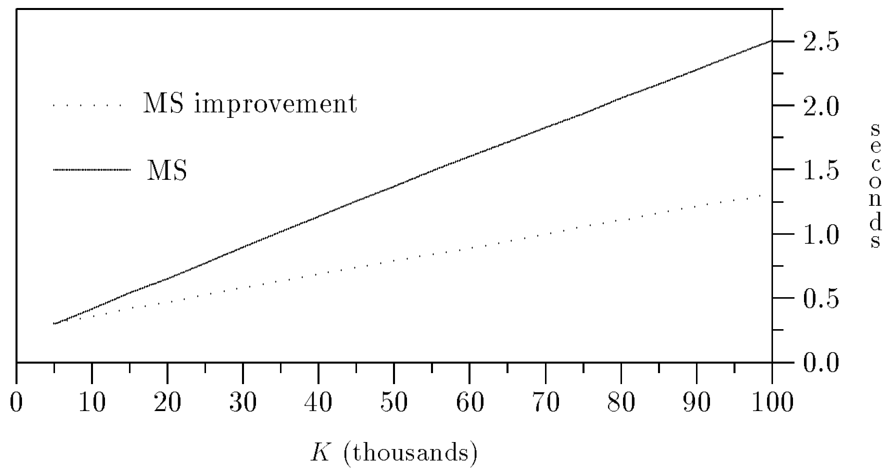

Figure 5: Random Networks, 10000 nodes and 150000 arcs

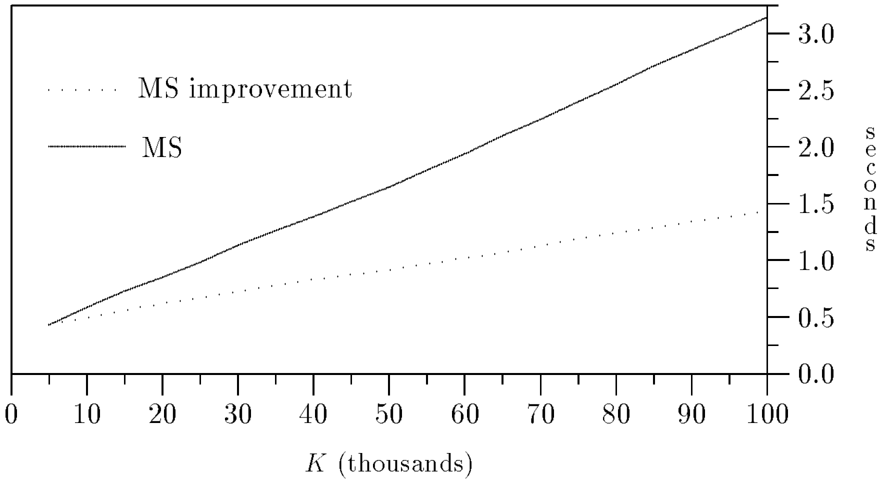

Figure 6: Random Networks, 10000 nodes and 200000 arcs

Acknowledgments:
Research developed within **CISUC** ("Centro de Informática e Sistemas da Universidade de Coimbra") and partly supported by the portuguese Ministry of Science and Technology (MCT), under **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. 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.

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

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

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

[8] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery*, 34:596–615, 1987.

[9] 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.

[10] 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).

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

[12] E. Q. V. Martins, M. M. B. Pascoal, D. M. L. D. Rasteiro and J.L.E. Santos. The optimal path problem. *Investigação Operacional*, 19:43–60, 1999. (http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/opath.ps.gz).

[13] E. Q. V. Martins, M. M. B. Pascoal and J. L. E. Santos. A new algorithm for ranking loopless paths. *Research Report, CISUC*, 1997. (http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/mps.ps.gz).

[14] E. Q. V. Martins, M. M. B. Pascoal and J. L. E. Santos. Labeling algorithms for ranking shortest paths. Submitted, 1998. (http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/labeling.ps.gz).

[15] 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).

[16] E. Q. V. Martins and J. L. E. Santos. A new shortest paths ranking algorithm. To appear in *Investigação Operacional* (http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/K.ps.gz).

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