Pré-Publicações do Departamento de Matemática
Universidade de Coimbra
Preprint Number 07–07

k-SHORTEST PATH ALGORITHMS

JOSÉ LUIS SANTOS

ABSTRACT: This paper focuses on algorithms to solve the k-shortest path problem. Three codes are described and compared on random generated and real-world networks. One million paths were ranked in less than 3 seconds (3 microseconds per path), with at most 1 second of preprocessing, on random generated networks with 10 000 nodes. For real-world instances with more than one million nodes, the preprocessing time rises up to 2,7 hours and the CPU time to rank one million paths is less than 30 seconds (30 microseconds per path).

KEYWORDS: shortest path, ranking path, deviation path.

1. Introduction

The shortest path problem was one of the first network problems studied in terms of operations research, [1, 7]. Fixed two specific nodes s and t in the network, the goal is to find a minimum cost way to go from s to t.

The first papers dealing with this subject appeared in the last years of the 1950s, [6, 10, 22]. In [9, 16], one can find an extensive bibliography of published papers about the shortest path problem until 1984.

The k-shortest path problem is a variant of the shortest path problem, where one intends to determine k paths $p_1, \ldots, p_k$ (in order), between two fixed nodes. Each path p_i should have cost greater or equal than p_{i-1}, $1 < i \leq k$, and the remainder paths between the fixed nodes should have cost at least equal to p_k. This problem has been well-studied, [4, 14, 15, 17, 19, 20, 21, 24, 25, 26, 27], and many algorithms are known. Dreyfus, [11], and Yen, [28] cite several additional papers on this subject going back as far as 1957 and Eppstein have an on-line bibliography, [12], with hundreds of references updated until 2001.

There are several applications on this problems applied to other network optimization problems. One of them is the restricted shortest path, where the shortest path that verifies a certain condition is searched. This problem

Received February 28, 2007.
The author gratefully acknowledges partial financial support by Centro de Matemática da Universidade de Coimbra and thanks José Paulo Almeida for the useful comments to improve the writing of this paper.

can be solved ranking paths until the first one which satisfies the condition given is found.

However, usually there is no upper bound for the number of paths to be ranked which constitutes a serious handicap for this method. Depending on the restriction given, one needs to rank less or more paths and it may be or not an efficient way to solve it. For instance, when the condition is the path which passes through all nodes only once (i.e. the shortest Hamiltonian path), ranking paths does not solve the problem for large networks. On the other hand, if one is interested on ranking loopless paths (i.e. paths without repeated nodes), this method produces very good results, [19].

This paper is divided into five sections. Sections 1 and 2 are devoted to a short background and the mathematical description of the k-shortest path problem, respectively. In section 3 we give a briefly description of the algorithms used. Finally, computational results and the conclusion are reported in sections 4 and 5, respectively.

2. Problem description

A network, G, is defined upon a set of nodes, $N = \{1, \ldots, n\}$, and a set of arcs, $A = \{a_1, \ldots, a_m\} \subset N \times N$. An arc links two nodes, i and j, in the network and mean that one can pass from one node (i or j) to the other. When the arc is oriented, we can only pass in one direction through this arc (it will be the case in this paper). So, an arc a_k can be represented by a pair of nodes, $a_k = (i, j)$, where i is called the tail and j the head node of the arc. We will denote by $A_i^+ = \{(j, i) : (j, i) \in A \text{ and } j \in N\}$ the set of arcs incoming to node i and by $A_i^- = \{(i, j) : (i, j) \in A \text{ and } j \in N\}$ the set of arcs outgoing node i.

Each arc $a_k = (i, j)$ has associated a value, c_{a_k} or $c_{i,j}$, indicating the cost (or distance, time, etc.) to cross the arc.

A path is a sequence of arcs where the head node of one arc is the tail node of the next arc in the sequence. If there are no multiple arcs (i.e. arcs with the same pair of tail and head nodes), a path only can be represented by the sequence of the nodes wherein the path passes through. For instance, path $p = \langle (v_0, v_1), (v_1, v_2), \ldots, (v_{\ell-1}, v_\ell) \rangle$ will be represented only by $p = \langle v_0, v_1, v_2, \ldots, v_{\ell-1}, v_\ell \rangle$. The cost of path p is the sum of the arcs cost in p and it will be denoted by $f(p)$. So, $f(p) = \sum_{(i,j)\in p} c_{i,j}$. We fixed two nodes in the network, the initial node (s) and the terminal one (t) and we denote by P the set of paths from node s to node t.

The shortest path problem consists of determining a path $p* \in P$ such that $f(p*) \leq f(q), \forall q \in P$. In a similar way, in the k-shortest path problem one is looking for k paths $(p_1, \ldots, p_k)$ verifying $f(p_i) \leq f(p_{i+1})$, $1 \leq i < k$, and $f(p_k) \leq f(q), \forall q \in P - \{p_1, \ldots, p_k\}$.

3. Algorithms description

In this section, the algorithms used in this work are described. In order to simplify this task, it is assumed that there are no arcs with s as head node nor t as tail node. If it is not the case, one can add two new nodes (S and T) to the network and the zero cost arcs (S, s) and (t, T). The initial and terminal nodes have to be redefined as S and T, respectively.

3.1. Removing path algorithm. This algorithm was proposed by Martins, [17, 18], in 1984. The main idea takes into account the following property: the second shortest path p_2 in G is the shortest path in a new network G', obtained from G removing the shortest path p_1. In addition, the third shortest path p_3 in G corresponds to the shortest path in a network G'' obtained from G removing p_1 and p_2, or removing p_2 from G'. Consequently, the general steps of the algorithm are:

- the removal of the shortest path in the current network;
- the determination of the shortest path in the resulting network.

A path $p = \langle s = v_0, v_1, \ldots, v_\ell = t\rangle$ can be removed from network G building a new network G' as follows:

- transfer the nodes and arcs of G to G';
- make a copy of path p, creating copies of the internals nodes of p, that is, $N' = N \cup \{v'_1, \ldots, v'_{\ell-1}\}$ (observe that v_0 is not copied; however, to simplify this description we will write v'_0 to represent v_0);
- join the arcs $\{(v'_{i-1}, v'_i)\}$, $1 < i < \ell$, to the new network G';
- link each internal node v'_i to the original network G. For each arc $(j, v_i) \in A^+_{v_i}$, with $i \neq v_{i-1}$ and $i \in \{1, \ldots, \ell - 1\}$, put a new arc $(j, v'_i) \in {A'}^+_{v'_i}$ with the same cost value. So, ${A'}^+_{v'_i} = \{(j, v'_i) : (j, v_i) \in A \text{ and } j \in N - \{v_{i-1}\}\} \cup \{(v'_{i-1}, v'_i)\}$;
- move the arc $(v_{\ell-1}, v_\ell)$ to $(v'_{\ell-1}, v_\ell)$ (consequently, $\langle v_0, v_1, \ldots, v_\ell\rangle$ is removed from G'). Thus, $A' = (A \backslash \{(v_{\ell-1}, v_\ell)\}) \cup \{(v'_{\ell-1}, v_\ell)\} \cup \bigcup_{i=1}^{\ell-1} {A'}^+_{v'_i}$.

Let T_s (*vs.* T'_s) be the shortest tree rooted at s in network G (*vs.* G'). Then, T_s (*vs.* T'_s) is formed by shortest paths from s to all nodes of N (*vs.*

N'). As there are no arcs from the new nodes v_i' to the original ones (except the arc $(v_{\ell-1}', t)$), then the labels of nodes in $N - \{t\}$ are kept from T_s to T_s'. On the other hand, the new nodes v_i' are labelled with $\pi_{v_i'} = \min\{\pi_j + c_{j,v_i'} : (j, v_i') \in A^+_{v_i'}\}$, $1 \leq i < \ell$, where π_j corresponds to the shortest path value from s to j in G. Hence, T_s' can be updated quickly at each iteration.

It can be proved, [18, 23], that the set of paths from s to t in G' corresponds to $P - \{p\}$. Note that, if $\#A^+_{v_1} = 1$, node v_1' will be redundant in G' because there are no arcs incoming to this node. The same happen to v_2' when $\#A^+_{v_1} = \#A^+_{v_2} = 1$. Consequently, some improvements were carried out in order to eliminate redundant nodes, [2, 3, 5, 4], leading to a version where the shortest path from s to v_i' corresponds to the next shortest path from s to v_i.

In 1995, Martins and Santos, [21, 23], noticed that $A^+_{v_i}$ is no longer used after the creation of node v_i'. As a consequence, the information in $A^+_{v_i'}$ can be stored in $A^+_{v_i}$, allowing to save a large amount of memory and CPU time.

The last improvement was produced in 1999, [20], sorting the tail nodes of $A^+_{v_i}$ by the value $\pi_j + c_{j,v_i}$, for all node j such that $(j, v_i) \in A^+_{v_i}$. So, the label of the node v_i' is obtained from the first arc of $A^+_{v_i}$. Recall that this sorting needs to be updated when a new copy of v_i is made.

3.2. Deviation path algorithm. This algorithm is based upon Eppstein's work, [13, 14], describing how to obtain the k-shortest path from deviation paths of $p_1, p_2, \ldots, p_{k-1}$. Thus, these deviation paths are candidates for the next shortest path, being p_k the one with minimum cost.

From now on, let us denote the shortest tree rooted at t by T_t and the shortest path from i to t in T_t by $T_t(i)$. The keyword of this algorithm is the computation of deviation paths from a path $p = \langle v_0, \ldots, v_\ell \rangle$ in the network. We say that $q = \langle u_0, \ldots, u_w \rangle$ is a shortest deviation path of p if there is $x \in \mathbb{N}_0$ such that

- $x < \ell$ and $x < w$;
- $v_i = u_i, 0 \leq i \leq x$;
- $v_{x+1} \neq u_{x+1}$;
- $\langle u_{x+1}, \ldots, u_w = t \rangle$ is the shortest path from u_{x+1} to t, that is, $T_t(u_{x+1})$.

In this case, we say (u_x, u_{x+1}) is the deviation arc and u_x the deviation node of q from p. We will denote p (the deviation path that produces q) by $\psi(q)$, u_x by $\phi(q)$ and (u_x, u_{x+1}) by $\theta(q)$.

To guarantee that each candidate is determined no more than once, it is imposed that deviations paths can only be generated from nodes following the deviation node.

The reduced cost of an arc (i, j) relatively to T_t, $\bar{c}_{i,j}$, makes easier the computation of the cost of deviation paths. Indeed, denoting by π_x the value of the shortest path from x to t, the reduced cost of arc (i, j) is defined by $\bar{c}_{i,j} = c_{i,j} + \pi_j - \pi_i$ and satisfies the following properties:

(1) $\bar{c}_{i,j} \geq 0$, $\forall (i, j) \in A$;
(2) $\bar{c}_{i,j} = 0$, $\forall (i, j) \in T_t$;
(3) $\bar{f}(q) = \sum_{i=1}^{w} \bar{c}_{u_{i-1},u_i} = \pi_t - \pi_s + \sum_{i=1}^{w} c_{u_{i-1},u_i} = \pi_t - \pi_s + f(q)$.

The last property assures that ranking paths by f value is identical from the rank obtained with $\bar{f}$ value. In addition, the second property tells us that if q is a deviation path from p with deviation arc (u_x, u_{x+1}), then $\bar{f}(q) = \bar{f}(p) + \bar{c}_{u_x,u_{x+1}}$.

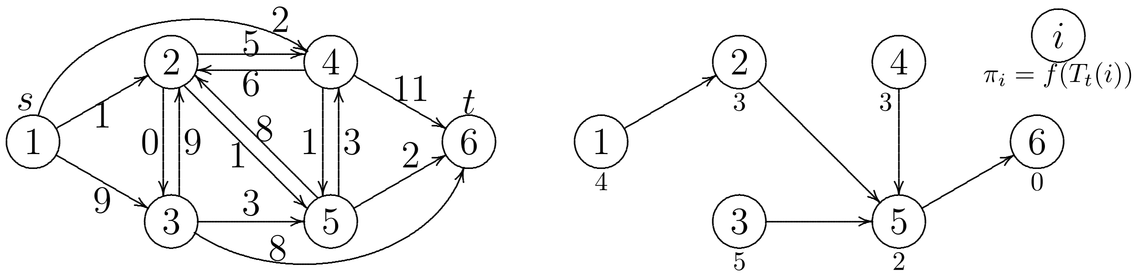
The deviation path algorithm starts with the determination of T_t, and put the shortest path from s to t in a set of candidates for the next shortest path denoted by X. So, initializing k with 0, the general steps of this algorithm consist of:

- $k = k + 1$;
- let p_k be the path with minimum $\bar{f}$ value in X;
- remove p_k from X;
- join all shortest deviation paths from p_k to X.

We can decrease the number of elements in X if we sort A_i^- by the value of the reduced cost, [19]. We would like to emphasize that the first arc of A_i^- must be the one belonging to T_t.

In this way, supposing that $p_k = \langle v_0, \ldots, v_\ell \rangle$ and $\phi(p_k) = v_y$ (for some $y \in \{1, \ldots, \ell - 1\}$), we have to compute a unique shortest deviation path for each node $v_x \in T_t(v_y) = \langle v_y, \ldots, v_\ell \rangle$, using the arc following (v_x, v_{x+1}) in $A_{v_x}^-$, $y \leq x < \ell$. Note that the candidate determined at node $\phi(p_k)$ is the next shortest deviation path from $\psi(p_k)$ at node $\phi(p_k)$.

3.3. Deviation path algorithm - a new improvement. In this paper, we propose a new variant for the deviation path algorithm, computing only two candidates for each path ranked. In fact, in the previous algorithm we compute one shortest deviation path for each node of p_k starting at its deviation node. However, only one of these candidates is needed at each



(a) G (b) T_t

FIGURE 1. (a) Network (G) to simulate the new algorithm. (b) The shortest tree rooted at $t = 6$ (T_t)

i	π_i		sub-set of arcs and reduced cost				
1	4	A_1^-	(1,2)	$\rightarrow$	(1,4)	$\rightarrow$	(1,3)
		$\bar{c}$	0		1		10
2	3	A_2^-	(2,5)	$\rightarrow$	(2,3)	$\rightarrow$	(2,4)
		$\bar{c}$	0		2		5
3	5	A_3^-	(3,5)	$\rightarrow$	(3,6)	$\rightarrow$	(3,2)
		$\bar{c}$	0		3		7
4	3	A_4^-	(4,5)	$\rightarrow$	(4,2)	$\rightarrow$	(4,6)
		$\bar{c}$	0		6		8
5	2	A_5^-	(5,6)	$\rightarrow$	(5,4)	$\rightarrow$	(5,2)
		$\bar{c}$	0		4		9

TABLE 1. Reduced cost and the shortest tree T_t for the network in Figure 1.

iteration. So, the shortest deviation paths of a generic path p should be sorted. This can be accomplished sorting the arcs of $\bigcup_{i \in T_t(y)} A_i^-$ by the reduced cost, where y is the head node of the deviation arc of p, that is, $\theta(p) = (\phi(p), y)$. We would like to emphasize that the sort "changes" with node y. To illustrate this, let us consider the network depicted in Figure 1. The reduced cost of the arcs for this example are reported in Table 1. When $y = 2$ we desire to sort $A_2^- \cup A_5^-$ and then, some arcs of A_2^- may appear in the middle of A_5^-. However, the arcs of A_2^- should be ignored for $y = 5$. In order to know which arcs are available for some value y, the sort must be restricted to the arcs which tail node belongs to $T_t(y)$.

So, in the new algorithm, the A is sorted by the reduced cost value (irrespective of its tail node), assuming that the first $n - 1$ arcs are the ones belonging to T_t. Let us denote by S the sequence of arcs after the sorting have been accomplished.

In the general step of the new algorithm, we pickup as p_k the minimum deviation path stored in X. This path produces two new candidates for p_{k+1} which are added to X:

- the shortest deviation path from p_k that uses the first arc $(u, v) \in S \backslash T_t$ verifying $u \in T_t(x)$, where x is the head node of the deviation arc of p_k (i.e. $\theta(p_k) = (\phi(p_k), x)$);
- the shortest deviation path from $\psi(p_k)$ that uses the first arc (w, z) of $S \backslash T_t$ after $\theta(p_k)$ verifying $w \in T_t(y)$, where y is the head node of the deviation arc of $\psi(p_k)$ (i.e. $\theta(\psi(p_k)) = (\phi(\psi(p_k)), y)$);

Due only two candidates are generated per each path ranked, the new algorithm is very promissory. If a proper data structure is used to get quickly the next arc in S, this algorithm will have a hight performance. Two data structure can be easily used:

- a simple list containing the sequence stored in S. This data structure is very efficient in terms of memory space, but it may be necessary to search the entire list to find the desire arc;
- a n-dimensional vector of list, putting the sublist of S with the elements of $\bigcup_{j \in T_t(i)} A_j^-$ in the i-th component of the vector.

Table 2 shows the above mentioned data structures for the network of Figure 1. The simulation of the new algorithm for this example to rank the first five shortest path is reported in Table 3. In this paper, we use a data structure similar to the simple list presented in Table 2, where additional links were added in order to shortcut the access to some elements in S.

4. Computational results

The computational experiment was carried out on a Intel(R) Pentium(R) 4 CPU 3.00GHz personal computer with 512 MB RAM and 1 MB cache size at the Laboratory for Computational Mathematics of Centre for Mathematics of the University of Coimbra. Codes was written in C language and compiled using the "cc" compiler of Linux system (Suse 9.3 version) without any optimization option.

In this section, we study the performance of the codes "rem", "dev" and "new", corresponding to implementations of the removing path, the deviation path and the new algorithms, respectively. The three codes for solving the simple list

(5,6)	$\rightarrow$	(4,5)	$\rightarrow$	(3,5)	$\rightarrow$	(2,5)	$\rightarrow$	(1,2)	$\rightarrow$	(1,4)	$\rightarrow$
	$\rightarrow$	(2,3)	$\rightarrow$	(3,6)	$\rightarrow$	(5,4)	$\rightarrow$	(2,4)	$\rightarrow$	(4,2)	$\rightarrow$
	$\rightarrow$	(3,2)	$\rightarrow$	(4,6)	$\rightarrow$	(5,2)	$\rightarrow$	(1,3)			

n-dimensional vector of list

node i	set of arc $\bigcup_{j\in T_t(i)} A_j^-$ ordered
1	(5,6) $\rightarrow$ (2,5) $\rightarrow$ (1,2) $\rightarrow$ (1,4) $\rightarrow$ (2,3) $\rightarrow$ (5,4) $\rightarrow$ (2,4) $\rightarrow$ (5,2) $\rightarrow$ (1,3)
2	(5,6) $\rightarrow$ (2,5) $\rightarrow$ (2,3) $\rightarrow$ (5,4) $\rightarrow$ (2,4) $\rightarrow$ (5,2)
3	(5,6) $\rightarrow$ (3,5) $\rightarrow$ (3,6) $\rightarrow$ (5,4) $\rightarrow$ (3,2) $\rightarrow$ (5,2)
4	(5,6) $\rightarrow$ (4,5) $\rightarrow$ (5,4) $\rightarrow$ (4,2) $\rightarrow$ (4,6) $\rightarrow$ (5,2)
5	(5,6) $\rightarrow$ (5,4) $\rightarrow$ (5,2)

TABLE 2. Two possible data structures to keep A order as requested in the new algorithm.

			new shortest deviation paths				
k	p_k	X	path q	$\psi(q)$	$\theta(q)$	$\phi(q)$	$\bar{f}(q)$
0	—	$\{q_1\}$	$q_1 = \langle 1,2,5,6\rangle$	the shortest $s-t$ path in T_t			
1	q_1	$\{q_2\}$	$q_2 = \langle 1,4,5,6\rangle$	q_1	(1, 4)	1	1
2	q_2	$\{q_3, q_4\}$	$q_3 = \langle 1,2,3,5,6\rangle$	q_1	(2, 3)	2	2
			$q_4 = \langle 1,4,5,4,5,6\rangle$	q_2	(5, 4)	5	5
3	q_3	$\{q_5, q_4, q_6\}$	$q_5 = \langle 1,2,5,4,5,6\rangle$	q_1	(5, 4)	5	4
			$q_6 = \langle 1,2,3,6\rangle$	q_3	(3, 6)	3	5
4	q_5	$\{q_4, q_6, q_7, q_8\}$	$q_7 = \langle 1,2,4,5,6\rangle$	q_1	(2, 4)	2	5
			$q_8 = \langle 1,2,5,4,5,4,5,6\rangle$	q_5	(5, 4)	5	8
5	q_4	$\{q_6, q_7, q_9, q_8, q_{10}\}$	$q_9 = \langle 1,4,2,5,6\rangle$	q_2	(4, 2)	4	7
			$q_{10} = \langle 1,4,5,4,5,4,5,6\rangle$	q_4	(5, 4)	5	9

TABLE 3. Simulation of the new algorithm for the network depicted in Figure 1.

k-shortest path problem were compared in two classes of problems: randomly generated and real-world instances. The CPU time was measured in seconds.

4.1. Randomly generated instances. In this section, two families of networks "rand" and "grid" were produced with the random network generators "sprand.exe" and "spgrid.exe", respectively, available online in [8]. For each network family, two kinds of arc cost were considered: randomly arc cost chosen in the interval $[1, 1000]$ ("rand-C" and "grid-C") and unitary arc cost ("rand-1" and "grid-1"). The presented values correspond to the average results for 1000 queries pairs. The performance of the algorithms is evaluated for different values of paths ranked (k) and for several values of the number of nodes (n) and the density (d = number of arcs/n) of the underlying graph.

Table 4 - 8 summarizes our computational experiment for randomly generated instances, determining one million path in less than 3 seconds of CPU time, after a preprocessing time less than 0,66 seconds. The computational results show that unitary arc cost instances are easier to solve than randomly arc cost for "rand" instances. However, the arc cost distribution seems not affect significatively the results on "grid" networks. Finally, "rem" algorithm has the lowest preprocessing time while "dev" is the fastest to do the rank. Although "new" to be the slowest code, it produces the smallest number of candidates for the next shortest path, allowing to solve the biggest problems (see Table 9).

The preprocessing time increases with n and d, being the results for the new algorithm in randomly arc cost instances the one where the preprocessing time enlarges quickly. The CPU time to accomplish the rank of the k shortest paths enhances with k and n, but decreases with d (except for the new algorithm where the time to rank one million paths also rises with d).

We would like to point out that "new" computes, at most, two candidates for each path ranked. On the other hand, the number of candidates created by "rem" and "dev" is not predictable (for the "rem" code, the number of candidates corresponds to the number of ranked shortest sub-paths determined). This value is affected by the number of paths to rank (k) only in grid networks. On the other hand, it increases with n (slightly in "rand" networks and faster in "grid" ones) and decreases with d (stressed results for unitary cost instances). Note that n can be taken as an upper bound for the number of candidates generated for each path ranked, but it is too large to be used.

4.2. Real-world instances. In this section, we use the "TIGER/Line" collection of real-world instances available for the "*9th DIMACS Implementation Challenge - Shortest Paths*", [8] which corresponds to the (undirected) road networks of the 50 US States and the District of Columbia (USA-road-d package).

rand-C networks

code	pp	$k =$	100	200	300	400	500	600	700	800	900	1000
rem	0,04	rank	0,30	0,57	0,81	1,04	1,26	1,48	1,69	1,90	2,11	2,31
		ratio	8,91	8,86	8,85	8,84	8,83	8,83	8,83	8,82	8,82	8,82
dev	0,05	rank	0,17	0,34	0,51	0,68	0,85	1,01	1,18	1,34	1,50	1,66
		ratio	9,81	9,81	9,81	9,81	9,81	9,81	9,81	9,81	9,81	9,81
new	0,45	rank	0,31	0,61	0,91	1,20	1,48	1,76	2,03	2,30	2,57	2,84
		ratio	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

rand-1 networks

code	pp	$k =$	100	200	300	400	500	600	700	800	900	1000
rem	0,01	rank	0,09	0,17	0,24	0,32	0,39	0,46	0,53	0,60	0,67	0,74
		ratio	4,76	4,77	4,77	4,76	4,73	4,70	4,69	4,68	4,67	4,67
dev	0,01	rank	0,08	0,16	0,25	0,33	0,42	0,50	0,59	0,67	0,76	0,84
		ratio	5,52	5,52	5,52	5,51	5,52	5,52	5,52	5,52	5,52	5,52
new	0,03	rank	0,12	0,24	0,36	0,48	0,60	0,73	0,85	0,97	1,09	1,21
		ratio	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

pp = preprocessing time (sec); rank = CPU time (sec) for ranking one million paths; ratio = average number of candidates per path ranked.

TABLE 4. Computational results for ranking k thousands paths in "rand" networks with 10000 nodes and 100000 arcs.

		rand-C networks					rand-1 networks				
code	n	2000	4000	6000	8000	10000	2000	4000	6000	8000	10000
rem	pp	0,00	0,00	0,01	0,03	0,04	0,00	0,00	0,00	0,01	0,01
	rank	1,42	1,78	1,98	2,17	2,31	0,60	0,66	0,69	0,71	0,74
	ratio	7,13	7,96	8,24	8,63	8,82	4,04	4,35	4,47	4,57	4,67
dev	pp	0,00	0,00	0,01	0,03	0,05	0,00	0,00	0,00	0,00	0,01
	rank	1,05	1,26	1,40	1,55	1,66	0,54	0,64	0,72	0,79	0,84
	ratio	8,12	8,96	9,23	9,62	9,81	4,82	5,11	5,28	5,42	5,52
new	pp	0,02	0,09	0,17	0,29	0,45	0,00	0,01	0,01	0,02	0,03
	rank	1,53	2,01	2,36	2,62	2,84	0,71	0,90	1,06	1,15	1,21
	ratio	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

pp = preprocessing time (sec); rank = CPU time (sec) for ranking one million paths; ratio = average number of candidates per path ranked.

TABLE 5. Computational results in "rand" networks with n nodes and $10n$ arcs.

The values presented in Table 9 correspond to the average results for some aleatory query pairs (see column "q"). We would like to emphasize that the "new" code is the only one which ranks one million paths in all the cases.

code	d	rand-C networks 2	4	6	8	10	rand-1 networks 2	4	6	8	10
	pp	0,00	0,01	0,02	0,03	0,04	0,00	0,00	0,01	0,01	0,01
rem	rank	2,87	2,61	2,43	2,35	2,28	1,47	1,06	0,87	0,79	0,74
	ratio	9,49	9,22	8,79	8,74	8,77	9,79	7,06	5,73	5,07	4,67
	pp	0,00	0,01	0,02	0,03	0,05	0,00	0,00	0,01	0,01	0,01
dev	rank	1,71	1,63	1,63	1,65	1,66	1,33	1,07	0,96	0,89	0,84
	ratio	11,89	10,30	9,80	9,78	9,81	10,67	7,74	6,53	5,90	5,52
	pp	0,00	0,03	0,07	0,17	0,45	0,00	0,01	0,01	0,02	0,03
new	rank	2,63	2,64	2,76	2,80	2,84	1,80	1,44	1,31	1,23	1,21
	ratio	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

pp = preprocessing time (sec); rank = CPU time (sec) for ranking one million paths; ratio = average number of candidates per path ranked.

TABLE 6. CPU time for ranking one million paths in rand networks with 10000 nodes and 10000d arcs.

grid-C networks

code	pp	$k =$	100	200	300	400	500	600	700	800	900	1000
rem	0,00	rank	0,26	0,51	0,75	0,99	1,22	1,45	1,68	1,91	2,13	2,34
		ratio	17,32	16,66	16,32	16,09	15,91	15,77	15,66	15,57	15,40	15,25
dev	0,02	rank	0,19	0,37	0,54	0,71	0,88	1,05	1,22	1,39	1,55	1,70
		ratio	18,21	17,61	17,28	17,05	16,89	16,75	16,64	16,55	16,38	16,20
new	0,66	rank	0,30	0,59	0,87	1,15	1,42	1,69	1,95	2,22	2,48	2,74
		ratio	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

grid-1 networks

code	pp	$k =$	100	200	300	400	500	600	700	800	900	1000
rem	0,00	rank	0,26	0,51	0,74	0,97	1,20	1,43	1,66	1,88	2,11	2,33
		ratio	16,17	15,57	15,21	15,00	14,84	14,71	14,61	14,51	14,44	14,37
dev	0,02	rank	0,18	0,35	0,53	0,70	0,86	1,03	1,19	1,35	1,52	1,68
		ratio	16,98	16,42	16,13	15,92	15,76	15,64	15,54	15,46	15,38	15,32
new	0,66	rank	0,26	0,50	0,73	0,96	1,19	1,41	1,63	1,85	2,07	2,28
		ratio	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

pp = preprocessing time (sec); rank = CPU time (sec) for ranking one million paths; ratio = average number of candidates per path ranked.

TABLE 7. Computational results for ranking k thousand paths in square grid networks with 10000 nodes.

"rem" and "dev" can rank one million paths just in a few percentage of instances (see column "%") and only for small values of k they find the k shortest paths in all the problems (column "k"). Consequently, the results presented in column "ratio" and "rank" are referred to the value of k presented in column k. The last column (k-new) indicates the CPU time used

		grid-C networks					grid-1 networks				
	ℓ	20	40	60	80	100	20	40	60	80	100
code	n	400	1600	3600	6400	10000	400	1600	3600	6400	10000
	pp	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
rem	rank	0,75	1,11	1,44	1,87	2,34	0,78	1,17	1,51	1,97	2,34
	ratio	5,02	7,42	9,66	12,48	15,25	5,02	7,42	9,66	12,48	14,42
	pp	0,00	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,00	0,02
dev	rank	0,66	0,91	1,13	1,41	1,72	0,68	0,96	1,20	1,49	1,68
	ratio	5,99	8,40	10,65	13,46	16,38	5,99	8,40	10,65	13,46	15,32
	pp	0,00	0,01	0,06	0,22	0,66	0,00	0,01	0,06	0,22	0,69
new	rank	0,81	1,15	1,58	2,17	2,74	0,83	1,20	1,64	2,25	2,63
	ratio	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

pp = preprocessing time (sec); rank = CPU time (sec) for ranking one million paths; ratio = average number of candidates per path ranked.

TABLE 8. Computational results for ranking one million paths in square grid networks with $n = \ell^2$ nodes.

by "new" algorithm to rank the same number of paths found by "rem" or "dev" code, respectively.

On this class of instances, the number of candidates generated per each path ranked is much greater than for the randomly generated instances (see column "ratio"). Consequently, "new" code outperforms "rem" (see column "k-new") in all instances and "dev" for the "NY" and "COL" network. However, "new" algorithm have an huge preprocessing time.

5. Conclusion

Three codes for solving the k-shortest path problem are described in this work. All of them are rather efficient, computing one million path in less than 3 seconds of CPU time in randomly generated networks (after .

The removing path algorithm has the lowest preprocessing time while deviation paths algorithm is the fastest to do the rank in randomly generated instances (where the number of deviation paths generated per each path ranked is small). The proposed algorithm have huge preprocessing time and is the slowest code in randomly generated instances. However, it produces the smallest number of candidates for the next shortest path (only two candidates), allowing to solve efficiently the biggest problems considered (real-world networks). In fact, it takes less than 30 seconds (CPU time) to rank one million paths in a real-world network with more than one million of nodes (after 2.7 hours of preprocessing).

Name	n	m	q	code	%	k	pp	rank	ratio	k-new
NY	264345	734610	50	"rem"	46,00	50	0,30	0,88	128,50	0,82
				"dev"	42,00	100	10,36	0,97	100,40	1,60
				"new"	100,0	1000	463,82	14,87	2,00	—
BAY	321269	801264	379	"rem"	30,87	50	0,65	0,62	126,14	0,39
				"dev"	33,77	50	6,74	0,42	103,05	0,39
				"new"	100,0	1000	260,42	7,10	2,00	—
COL	435665	1059582	100	"rem"	23,00	<50	0,75	1,15	240,17	0,50
				"dev"	28,00	100	16,45	1,15	275,92	0,98
				"new"	100,0	1000	553,90	9,53	2,00	—
FLA	1070375	2720400	5	"rem"	0,00	<50	5,55	0,55	118,56	1,07
				"dev"	20,00	300	124,96	3,33	105,28	5,68
				"new"	100,0	1000	9498,5	27,66	2,00	—

n: number of nodes; m: number of arcs; q: number os pairs queries used; %: percentage of instances for which one million paths were ranked; k: last values of k (thousands) for which all instances were solved; pp: preprocessing time (sec); rank: CPU time (sec) for ranking k thousands shortest paths; ratio = average number of candidates for the first k paths ranked.

TABLE 9. Computational results on USA road network.

Future work on this subject will focus on a more appropriated data structure in order to sort the set of arcs in a more efficient manner leading to the reduction of the preprocessing time.

References

[1] R.K. Ahuja, T. L. Magnanti, and J.B. Orlin. *Network Flows – theory, algorithms, and applications.* Prentice-Hall, Inc., New Jersey, 1993.

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

[3] J.A. Azevedo, J.J. Madeira, E.Q. Martins, and F.M. 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.

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

[5] J.A. Azevedo and E.Q. Martins. An algorithm for the multiobjective shortest path problem on acyclic networks. *Investigação Operacional*, 11 (1):52–69, 1991.

[6] R.E. Bellman. On a routing problem. *Quarterly Applied Mathematics*, 16:87–90, 1958.

[7] B. Borchers. A reading list in combinatorial optimization. (http://www.nmt.edu/ borchers/readings/coptreadings.ps), 1994.

[8] Dimacs Center. 9th dimacs implementation challenge - shortest paths, 2006. (http://www.dis.uniroma1.it/∼challenge9/).

[9] N. Deo and C. Pang. Shortest path algorithms: taxonomy and annotation. *Networks*, 14 (2):275–323, 1984.

[10] E. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:395–412, 1959.
[11] S. Dreyfus. An appraisal of some shortest paths algorithms. *Operations Research*, 17:395–412, 1969.
[12] D. Eppstein. Bibliography on ranking shortest paths. (http://liinwww.ira.uka.de/bibliography/Theory/k-path.html).
[13] David Eppstein. Finding the k shortest paths. In *35th IEEE Symp. Foundations of Comp. Sci.*, pages 154–165, 1994. Santa Fé.
[14] David Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.
[15] David Eppstein. Finding the k shortest paths, 1998. Department of Information and Computer Science, University of California, Irvine, CA 92717, Tech. Report 94-26.
[16] B. Golden and T. Magnanti. Deterministic network optimization: a bibliography. *Networks*, 7:149–183, 1977.
[17] E.Q. Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18:123–130, 1984.
[18] E.Q. Martins. *Determinação de Caminhos Óptimos em Redes Orientadas*. PhD thesis, Departamento de Matemática, Universidade de Coimbra, 1984.
[19] E.Q. Martins, M.M. Pascoal, and J.L. Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10 (3):247–261, 1999.
[20] E.Q. Martins, M.M. Pascoal, and J.L. Santos. A new improvement for a k shortest paths algorithm. *Investigação Operacional*, 21 (1):47–60, 2001.
[21] E.Q. Martins and J.L. Santos. A new shortest paths ranking algorithm. *Investigação Operacional*, 20 (1):47–62, 2000.
[22] E.F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.
[23] J.L. Santos. Determinação ordenada dos k trajectos mais curtos. Relatório de Estágio, Departamento de Matemática, Universidade de Coimbra, 1995.
[24] D. Shier. Interactive methods for determining the k shortest paths in a network. *Networks*, 6:151–159, 1976.
[25] D. Shier. On algorithms for finding the k shortest paths in a network. *Networks*, 9:195–214, 1979.
[26] J.Y. Yen. Shortest path network problems. (Mathematical Systems in Economics, Heft 18), Hain, (1975), Meisennheim am Glan.
[27] J.Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.
[28] J.Y. Yen. Another algorithm for finding the k shortest-loopless network paths. In *In 41st Mtg. Operations Research Society of America*, volume 20, page B/185, 1972.

José Luis Santos
Departamento de Matemática, Universidade de Coimbra, 3001-454 Coimbra, PORTUGAL
E-mail address: zeluis@mat.uc.pt
URL: http://www.mat.uc.pt/∼zeluis