K shortest path algorithms

José L. Santos
zeluis@mat.uc.pt
Centre for Mathematics, University of Coimbra
Department of Mathematics, University of Coimbra

August, 2006

Abstract

This paper is focussed on algorithms to solve the k-shortest path problem. Three codes were described and compared on rand and grid networks using random generators available on

http://www.dis.uniroma1.it/∼challenge9

Codes were also tested on the USA road networks available on the same URL address. One million paths were ranked in less than 3 seconds on random instances with 10 000 nodes and 10 seconds for real-world instances.

keywords: shortest path, ranking path, deviation path.

Corresponding author:
José Luis Esteves dos Santos (*zeluis@mat.uc.pt*).
Departamento de Matemática, Universidade de Coimbra,
Apartado 3008, 3001-454 Coimbra, Portugal.
Fax number: 00 351 239832568

1. Introduction

The shortest path problem was one of the first network problems studied in terms of operations research, [6]. 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 appear in the last years of 50th decade, [5, 8, 19]. In [7, 13], one can find an extensive bibliography of published paper 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. It has been well-studied, [3, 11, 12, 14, 16, 17, 18, 21, 22, 23, 24], and many algorithms are known. Dreyfus, [9], and Yen, [25] cite several additional papers on this subject going back as far as 1957. The interested reader may adresses to a bibliography due to Eppstein with hundred of references that is online with the following web page: http://liinwww.ira.uka.de/bibliography/Theory/k-path.html.

This problem has several applications in others network optimization problems. One of them is the restricted shortest path, where the shortest path that verifies a specified condition is searched. This problem can be solved ranking paths until find the first one which satisfies the condition given.

However, usually there is no an upper bound for the number of paths to be ranked and it is a seriously 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 passes for all nodes once (find 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 (paths without repeated nodes), this method produces very good results, [16].

The paper is divided into five sections. Section 2 is devoted to the mathematical description of the k-shortest path problem. In section 3 is given a briefly description of the algorithms used. Finally, section 4 and 5, computational results and the conclusion are reported.

2. Problem description

A network, $\mathcal{G}$, is defined upon a set of nodes, $\mathcal{N} = \{1, \ldots, n\}$, and a set of arcs, $\mathcal{A} = \{a_1, \ldots, a_m\}$. 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 $\mathcal{A}_i^+ = \{(j, i) : (j, i) \in \mathcal{A} \text{ and } j \in \mathcal{N}\}$ the set of arcs incoming to node i and $\mathcal{A}_i^- = \{(i, j) : (i, j) \in \mathcal{A} \text{ and } j \in \mathcal{N}\}$ the set of arcs outgoing node i.

We fixed two nodes in the network, the initial node (s) and the terminal one (t).

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 is no multiple arcs (arcs with the same tail and head nodes), a path can be represented only 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 $p = \langle v_0, v_1, v_2, \ldots, v_{\ell-1}, v_\ell \rangle$. Let us denote by $\mathcal{P}$ the set of paths from node s to node t. The cost of path p is the sum of the arcs cost in p and it will be denoted by $f(p)$. So, $f(p) = \sum_{(i,j) \in p} c_{i,j}$.

The shortest path problem consists of determining a path $p* \in \mathcal{P}$ such that $f(p*) \leq f(q), \forall q \in \mathcal{P}$. 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 $f(p_k) \leq f(q), \forall q \in \mathcal{P} - \{p_1, \ldots, p_k\}$.

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 is no arcs with s as head node neither 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.

3.1. Removing path algorithm

This algorithm was proposed by Martins, [14, 15], in 1984. The main idea takes into account the following property: the second shortest path p_2 in $\mathcal{G}$ is the shortest path in a new network $\mathcal{G}'$, obtained from $\mathcal{G}$ removing the shortest path p_1. In addition, the third shortest path p_3 in $\mathcal{G}$ corresponds to the shortest path in a network $\mathcal{G}''$ obtained from $\mathcal{G}$ removing p_1 and p_2, or removing p_2 from $\mathcal{G}'$. Consequently, the general steps of the algorithm are:

- remove the shortest path in the current network;
- determining the shortest path in the resulting network.

The remotion of a path $p = \langle s = v_0, v_1, \ldots, v_\ell = t\rangle$ in network $\mathcal{G}$ can be done building a new network $\mathcal{G}'$ from $\mathcal{G}$ as follows:

- make a copy of path p, creating copies of the internals nodes of p, that is, $\mathcal{N}' = \mathcal{N} \cup \{v'_1, \ldots, v'_{\ell-1}\}$ (observe that v_0 is not copied; however, to simplified 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 $\mathcal{G}'$;
- link each internal node v'_i to the original network $\mathcal{G}$. Do this replacing by v'_i the head node of the arcs incoming to v_i which are not in p, that is, $\mathcal{A}'^{+}_{v'_i} = \{(j, v'_i) : (j, v_i) \in \mathcal{A} \text{ and } j \in \mathcal{N} - \{v_{i-1}\}\} \cup \{(v'_{i-1}, v'_i)\}$ and $\mathcal{A}'^{+}_{v_i} = \{(v_{i-1}, v_i)\}$, $1 \leq i < \ell$.
- 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 $\mathcal{G}'$);

Let $\mathcal{T}_s$ ($\mathcal{T}'_s$) be the shortest tree rooted at s in network $\mathcal{G}$ ($\mathcal{G}'$), that is, a tree formed by shortest paths from s to all nodes of $\mathcal{N}$ ($\mathcal{N}'$). If $\mathcal{T}_s$ was already computed, then $\mathcal{T}'_s$

is obtained in a very easy way. In fact, note that labels of nodes in $\mathcal{N} - \{t\}$ are kept in $\mathcal{T}'_s$. On the other hand, the new nodes v'_i are labelled with $\pi_{v'_i} = \min\{\pi_j + c_{j,v'_i} : (j, v'_i) \in \mathcal{A}^+_{v'_i}\}$, $1 \leq i < \ell$, where π_j corresponds to the shortest path value from s to j in $\mathcal{G}$.

It can be proved, [20], that the set of paths from s to t in $\mathcal{G}'$ corresponds to $\mathcal{P} - \{p\}$. Note that, if $\#\mathcal{A}^+_{v_1} = 1$, node v'_1 will be redundant in $\mathcal{G}'$ because there is no arcs incoming to this node. The same happen to v'_2 when $\#\mathcal{A}^+_{v_1} = \#\mathcal{A}^+_{v_2} = 1$. Consequently, some improvements were done in order to eliminate redundant nodes, [1, 2, 4, 3], 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, [20, 18], notice that $\mathcal{A}^+_{v_i}$ is not longer used after the creation of node v'_i. As a consequence, the information in $\mathcal{A}^+_{v'_i}$ can be stored in $\mathcal{A}^+_{v_i}$, allowing to save a large amount of memory and CPU time.

The last improvement in the algorithm was produced in 1999, [17], sorting the tail nodes of $\mathcal{A}^+_{v_i}$ by the value $\pi_j + c_{j,v_i}$, for all node j such that $(j, v_i) \in \mathcal{A}^+_{v_i}$. So, the label of node v'_i is obtained from the first arc of $\mathcal{A}^+_{v_i}$. Note that we have to update this sorting when a new copy of v_i is made.

3.2. Deviation path algorithm

This algorithm starts with a work of Eppstein, [11], 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 $\mathcal{T}_t$ and the shortest path from i to t in $\mathcal{T}_t$ by $\mathcal{T}_t(i)$. 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 deviation path of p if there is $x \in \mathbb{N}_0$ such that

- $x < \ell$ and $x < w$;
- $v_i = u_i, 0 \leq i \leq x$;
- $v_{x+1} \neq u_{x+1}$;

- $\langle u_{x+1}, \dots, u_w = t \rangle$ is the shortest path from u_{x+1} to t, that is, $\mathcal{T}_t(u_{x+1})$.

In this case, we say (u_x, u_{x+1}) is the deviation arc of q from p.

To guarantee each candidate is determined no more than once, it is imposed that deviations paths can only be generated from nodes v_x of p in the last part of p, that is, from nodes v_x for which $\langle v_x, \dots, v_\ell = t \rangle \in \mathcal{T}_t$.

The reduced cost of an arc (i, j) relatively to $\mathcal{T}_t$, $\bar{c}_{i,j}$, makes easier the computation of the cost of deviation paths. In fact, 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. $\bar{c}_{i,j} \geq 0, \forall (i, j) \in \mathcal{A}$;
2. $\bar{c}_{i,j} = 0, \forall (i, j) \in \mathcal{T}_t$;
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 $\bar{f}(q) = \bar{f}(p) + \bar{c}_{u_x,u_{x+1}}$, where q is a deviation path from p with deviation arc (u_x, u_{x+1}).

The deviation path algorithm starts with the determination of $\mathcal{T}_t$, 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 deviation paths from p_k to X.

We can decrease the number of candidates determined if we sort $\mathcal{A}_i^-$ by the value of the reduced cost, [16]. We would like to emphasize that the arc of $\mathcal{A}_i^-$ belonging to $\mathcal{T}_t$ have to be the first one.

In this way, denoting by v_y the tail node of the deviation arc of p, we have only to compute one deviation path for each node v_x of $\langle v_y, \dots, v_\ell \rangle$ (that is $\mathcal{T}_t(v_y)$), using the arc following (v_x, v_{x+1}) in $\mathcal{A}^-_{v_x}$, $y \leq x < \ell$.

3.3. Deviation path algorithm - second version

In this paper, we propose a new variant for the deviation path algorithm, computing only two candidates for each path ranked. It consists of sorting the entire set of arcs $\mathcal{A}$ by the reduced cost value (independent its tail node), assuming that the first $n-1$ arcs as the ones belonging to $\mathcal{T}_t$. Let us denoted by $a_{\mathcal{T}_t}(i)$ the arc of $\mathcal{T}_t$ with i as its tail node. Therefore, supposing that p_k was obtained from p_z $(z < k)$ with the deviation arc (v_x, v_{x+1}), we have to add into X

- one candidate from p_k: the deviation path of p_k where its deviation arc is the first one following $a_{\mathcal{T}_t}(v_x)$ with tail node belonging to $\mathcal{T}_t(v_x)$;
- one candidate from p_z: the deviation path of p_z using as deviation arc the first one that follows (v_x, v_{x+1}) and its tail node belongs to $\mathcal{T}_t(u_r)$, where u_r is the tail node of the deviation arc of p_z.

Using a correct data structure, these arcs can be obtained efficiently.

4. Computational results

The computational experience was carried on a Intel(R) Pentium(R) 4 CPU 3.00GHz personal computer with 1024 KB cache size. Codes was written in C language and compiled using the ”cc” compiler of Linux system (Suse 9.3 version).

In this section, we compared the following codes:

- ”rem”: an implementation of the removing path algorithm;
- ”dev”: an implementation of the deviation path algorithm;
- ”dev2”: an implementation of the deviation path algorithm which computes only two candidates for each ranked path;

The three codes for solving the k-shortest path problem were compared in two kind of problems: random generated and real-world instances. The CPU time was measured in seconds with the "timer" function available in the web page

http://www.dis.uniroma1.it/∼challenge9

4.1. Random generated instances

In this case, instances were produced with the random generators "sprand.exe" and "spgrid.exe" available online on the URL address above mentioned. The arc cost was randomly chosen in the interval $[1, 1000]$. It was considered just a query pair for each graph. The initial and terminal node in rand instances were always 1 and n (number of nodes), respectively. For square grid graphs, they corresponded to nodes on corners of the grid. The computational results presented in this section correspond to the average of 30 instances of each type of network using the seed from 1 to 30 (it was considering only the first query pair of each instance). The code performance was evaluated changing the number of paths ranked (k), the number of nodes (n) and the graph density ($d =$ (number of arcs)$/n$).

Tables 1 – 3 and Tables 4 – 5 resume our computational experience on rand and grid instances, respectively, determining one million path in less than 3 seconds of CPU time.

Let us note that "dev2" needs an extra CPU time (see Table 6) to build the data structures which allows sorting the set of arcs before starting the ranking paths.

Finally, we would like to notice that "dev2" computes, at most, two candidates for each new ranked path. On the other hand, the number of candidates generated by "dev" is not predictable, Table 7. Taking into account our computational results, "dev" determines an average of 10 candidates for each new ranked path in rand networks and this ratio increases upon to 30, when grid networks are considered. In what concerns to "rem" codes, the number of ranked shortest paths to internal nodes is equivalent to the number of candidates generated by "dev" and these numbers are similar in rand and grid networks.

k	100	200	300	400	500	600	700	800	900	1000
rem	0,24	0,46	0,67	0,88	1,07	1,24	1,42	1,59	1,75	1,92
dev	0,14	0,29	0,44	0,59	0,74	0,89	1,03	1,17	1,32	1,46
dev2	0,26	0,53	0,80	1,06	1,32	1,57	1,81	2,06	2,30	2,54

Table 1: CPU time for ranking k thousand paths in rand networks with 10000 nodes and 100000 arcs.

n	2000	4000	6000	8000	10000
rem	1,14	1,42	1,61	1,86	2,02
dev	0,93	1,10	1,21	1,35	1,43
dev2	1,35	1,78	2,08	2,30	2,42

Table 2: CPU time for ranking one million paths in rand networks with n nodes and $10n$ arcs.

d	2	4	6	8	10
rem	2,95	2,08	2,14	3,27	2,10
dev	1,42	1,32	1,34	1,37	1,39
dev2	1,90	2,09	2,22	2,28	2,30

Table 3: CPU time for ranking one million paths in rand networks with 10000 nodes and $10000d$ arcs.

k	100	200	300	400	500	600	700	800	900	1000
rem	0,36	0,68	0,99	1,30	1,60	1,91	2,22	2,51	2,81	3,10
dev	0,43	0,70	0,96	1,22	1,35	1,62	1,88	2,16	2,43	2,70
dev2	0,33	0,54	0,75	0,95	1,05	1,24	1,44	1,63	1,82	2,01

Table 4: CPU time for ranking k thousand paths in square grid networks with 10000 nodes.

$x = y$	20	40	60	80	100
rem	1,16	2,48	2,65	2,69	3,35
dev	0,69	1,25	1,56	2,14	2,66
dev2	0,52	0,74	0,99	1,54	2,03

Table 5: CPU time for ranking one million paths in square grid networks with $x \times y$ nodes.

n	2000	4000	6000	8000	10000
rand	0,01	0,05	0,10	0,15	0,21

d	2	4	6	8	10
rand	0	0,01	0,04	0,11	0,21

$x = y$	20	40	60	80	100
grid	0	0,01	0,07	0,25	0,68

Table 6: CPU time to buil the data structure in "dev2" code.

rand

n	2000	4000	6000	8000	10000
rem	7,12	7,75	8,07	8,53	8,72
dev	8,12	8,75	9,06	9,52	9,71
dev2	2,00	2,00	2,00	2,00	2,00

rand

d	2	4	6	8	10
rem	13,71	9,46	8,83	8,85	8,72
dev	11,58	10,27	9,80	9,84	9,71
dev2	2,00	2,00	2,00	2,00	2,00

grid

$x = y$	20	40	60	80	100
rem	6,74	13,48	17,19	22,82	27,68
dev	7,69	14,48	18,18	23,81	28,64
dev2	2,00	2,00	2,00	2,00	2,00

Table 7: Average number of candidates generated by each path ranked.

	Name	NY	BAY	COL
	n	264345	321269	435665
	m	734610	801264	1059582
	$k = 1$	5,27	5,97	14.65
"rem"	ord	–	–	–
	K	50000	< 50000	50000
	rank	2,24	2,30	2,30
	ratio	37,69	12,00	27,45
"dev"	ord	0,04	0,04	0,06
	K	250000	150000	100000
	rank	0,69	1,04	1,17
	ratio	25,38	25,99	24,30
"dev2"	ord	157,78	263,39	539,25
	rank	7,29	6,95	9,53
	ratio	2,00	2,00	2,00

Table 8: Computational results on USA road network.

4.2. Real-world instances instances

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

The values presented in Table 8 correspond to the average results for the first 100 query pairs in file "problem.p2p" for each instance. Line "K" indicates the largest value of K for which the codes solve all the instances ("dev2" ranked always 1000000 paths). Finally, line "rank" reports the ratio CPU time between the code "rem" or "dev" and "dev2" for the number of paths pointed in line "K"; for "dev2" codes, it shows the CPU time for the largest values of K considered. From these results, one can conclude that "dev2" are the unique code which ranks one million paths in all instances. So, despite the great time to build the data structure, "dev2" have the best performance to rank path in real-world problems.

5. Conclusion

Three codes for solving the k-shortest path problem were described in this work. All of them are very efficient, computing one million path in less than 3 seconds of CPU time in random instances. The removing path have, in general, worst performance than the deviation path algorithm. On the other hand, the second variant of the deviation path algorithm takes advantage over the original one in grid networks.

In what concerns to real-world problems, "dev2" was the unique code capable to solve all instances.

Our future work on this subject is focussed on the data structure that allows to sort the set of arcs in the variant of the deviation path algorithm.

1. 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.
2. 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.
3. 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.
4. 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.
5. R.E. Bellman. On a routing problem. *Quarterly Applied Mathematics*, 16:87–90, 1958.
6. B. Borchers. A reading list in combinatorial optimization. (http://www.nmt.edu/ borchers/readings/coptreadings.ps), 1994.
7. N. Deo and C. Pang. Shortest path algorithms: taxonomy and annotation. *Networks*, 14 (2):275–323, 1984.
8. E. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:395–412, 1959.
9. S. Dreyfus. An appraisal of some shortest paths algorithms. *Operations Research*, 17:395–412, 1969.
10. David Eppstein. Finding the k shortest paths. In *35th IEEE Symp. Foundations of Comp. Sci.*, pages 154–165, 1994. Santa Fé.
11. David Eppstein. Finding the k shortest paths, 1998. Department of Information and Computer Science, University of California, Irvine, CA 92717, Tech. Report 94-26.
12. David Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673,

1998.

13. B. Golden and T. Magnanti. Deterministic network optimization: a bibliography. *Networks*, 7:149–183, 1977.
14. E.Q. Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18:123–130, 1984.
15. E.Q. Martins. *Determinação de Caminhos Óptimos em Redes Orientadas*. PhD thesis, Departamento de Matemática, Universidade de Coimbra, 1984.
16. 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.
17. 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.
18. E.Q. Martins and J.L. Santos. A new shortest paths ranking algorithm. *Investigação Operacional*, 20 (1):47–62, 2000.
19. 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.
20. J.L. Santos. Determinação ordenada dos k trajectos mais curtos. Relatório de Estágio, Departamento de Matemática, Universidade de Coimbra, 1995.
21. D. Shier. Interactive methods for determining the k shortest paths in a network. *Networks*, 6:151–159, 1976.
22. D. Shier. On algorithms for finding the k shortest paths in a network. *Networks*, 9:195–214, 1979.
23. J.Y. Yen. Shortest path network problems. (Mathematical Systems in Economics, Heft 18), Hain, (1975), Meisennheim am Glan.
24. J.Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.
25. J.Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.