A NEW ALGORITHM FOR RANKING LOOPLESS PATHS

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

{*eqvm,marta,zeluis*}*@mat.uc.pt*
Departamento de Matemática
Universidade de Coimbra
PORTUGAL

May 1997

Abstract: In this paper we present a new algorithm for ranking loopless paths which solves large problems. Computational experiments show us that the proposed algorithm ranks more than $500,000$ loopless paths in an euclidian network with $10,000$ nodes and $100,000$ arcs and using about 100 Mbytes of RAM, in about 0.35 seconds of CPU execution time for undirected networks and in about 0.15 seconds of CPU execution time for directed networks.

Some comparative computational experiments are also reported.

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

1 Introduction

Let $(\mathcal{N},\mathcal{A})$ denote a given network, where $\mathcal{N} = \{v_1, \ldots, v_n\}$ is a finite set whose elements are called nodes and $\mathcal{A} = \{a_1, \ldots, a_m\}$ is a proper sub–set of $\mathcal{N} \times \mathcal{N}$ whose elements are called arcs. Each arc a_k is a pair (v_i, v_j) where $v_i \neq v_j$, being v_i called the tail node of a_k and v_j its head node. If (v_i, v_j) is an ordered (unordered) pair, for any $(v_i, v_j) \in \mathcal{A}$, then $(\mathcal{N},\mathcal{A})$ is said to be a directed (undirected) network.

Let s, t be two given nodes of $(\mathcal{N},\mathcal{A})$. A path p from s to t in $(\mathcal{N},\mathcal{A})$ is a sequence of nodes and arcs of the form $p = \langle v'_1, a'_1, v'_2 \ldots, v'_h, a'_h, v'_{h+1}\rangle$, in which $v'_1 = s$ and $v'_{h+1} = t$, $v'_i \in \mathcal{N}$ for any $i \in \{1, \ldots, h+1\}$ and, for any $k \in \{1, \ldots, h\}$, $a'_k \in \mathcal{A}$ and $a'_k = (v'_k, v'_{k+1})$.

A loopless path is a sequence p where:

- $v'_1 \neq v'_i$ and $v'_{h+1} \neq v'_i$, for any $i \in \{2, \ldots, h\}$,
- $v'_i \neq v'_j$, for any $i, j \in \{2, \ldots, h\}$, such that $i \neq j$.

A cycle is a loopless path from some node to itself.

For any arc $a_k = (v_i, v_j) \in \mathcal{A}$, let $c_{a_k} \in \mathrm{I\!R}$ or $c_{ij} \in \mathrm{I\!R}$ denote a real number associated with $a_k = (v_i, v_j)$, and let $\mathcal{P}_{ij}$ denote the set of paths from i to j in $(\mathcal{N},\mathcal{A})$, for any $i, j \in \mathcal{N}$. To simplify the notation, $\mathcal{P}$ will be used to denote $\mathcal{P}_{st}$.

Let $c : \mathcal{P} \longrightarrow \mathbb{R} : p \longrightarrow c(p) = \sum_p c_{ij}$ be a real function defined over $\mathcal{P}$, which assigns a value to each path from s to t in $(\mathcal{N},\mathcal{A})$.

In the ranking paths problem, for a given positive integer K, it is intended to determine a set $\mathcal{P}_K = \{p_1, \ldots, p_K\} \subset \mathcal{P}$, such as:

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

This is a classical problem which can be divided into two classes: the unconstrained problem and the constrained one. As opposed to the unconstrained problem, in the constrained ranking paths problem all the paths have to satisfy some condition; for instance, to be loopless.

The unconstrained ranking paths problem is considered easier than the ranking loopless paths problem and very efficient algorithms exist to solve it. The interested reader is adressed to [1], [2], [6], [7] and to a complete online bibliography due to Eppstein (WWW adress – "http://liinwww.ira.uka.de/bibliography/Theory/k-path.html").

For the ranking loopless paths problem, we point out the classical Yen's algorithm, [5] and [8], which is valid for directed and undirected networks and an algorithm due to Katoh, Ibaraki and Mine, [4], which is only valid for undirected networks.

In this paper, a new algorithm for ranking loopless paths is proposed, which is also valid for either directed or undirected networks. Computational experiments, comparing this new algorithm with the classical Yen's algorithm and Katoh *et al.* algorithm are reported, as well as computational experiments showing some performances of the algorithm in the resolution of large problems.

Throughout it is assumed that $\sum_C c_{ij} \geq 0$ for any cycle C in directed networks, and $c_{ij} \geq 0$ for any arc (v_i, v_j) in undirected networks.

The reader is assumed to be familiar with the K shortest paths problem terminology.

2 The Algorithm

The algorithm uses the change of variable proposed by Eppstein in its algorithm for the unconstrained problem. This change of variable is stated in lemma 2.1.

Lemma 2.1 *Let π_i denote the shortest distance from v_i to t and let $\bar{c}_{ij} = \pi_j - \pi_i + c_{ij}, \forall (v_i, v_j) \in \mathcal{A}$. So, $\bar{c}(p) = \sum_p \bar{c}_{ij} = \pi_t - \pi_s + c(p)$ holds for any $p \in \mathcal{P}$.*

Proof: For any path p, only π_s and π_t appear exactly once on $\bar{c}(p) = \sum_p (\pi_j - \pi_i + c_{ij})$. For all the intermediate nodes v_j of p, if any, π_j appear exactly twice – once when the arc (v_i, v_j) is considered and other for (v_j, v_l), the next arc in path p. So, 1 and -1 are the coefficient of π_j in the sum and the lemma follows. ∎

The following theorems are immediate consequences of lemma 2.1, and presented with no proofs.

Theorem 2.1 *Let $p, q \in \mathcal{P}_{ij}$, for any $i, j \in \mathcal{N}$. Then $c(p) \leq c(q)$ if and only if $c(\bar{p}) \leq c(\bar{q})$.*

Theorem 2.2 *Let p_{it} be a shortest path from i to t in $(\mathcal{N},\mathcal{A})$ So, $\bar{c}(p_{it}) = 0$, for any $i \in \mathcal{N}$.*

The problem of the shortest tree that is determined by the shortest path from each node $i \in \mathcal{N} - \{t\}$ to t in $(\mathcal{N},\mathcal{A})$ is also assumed to be known; among the hundreds of references, the reader may adress [3] for details.

In the algorithm the set of arcs emerging from each node v_i has to be sorted by increasing value of $\bar{c}_{ij}$ and in such a way that the first emerging arc belongs to the shortest path from v_i to t.

The concept of deviation node of a path is used in the algorithm.

For all $k > 1$, $p_k \in \mathcal{P}_K - \{p_1\}$ is coincident with paths $p_1, \ldots, p_{k-1}$ from node s to nodes $v(p_1), \ldots, v(p_{k-1})$, respectively. For $i \in \{1, \ldots, k-1\}$, let $p_{s,v(p_i)}$ be the sub–path of p_k from s to $v(p_i)$ and let $p_{s,v(p_r)}$ denote the sub–path with largest number of nodes. Node $v(p_r)$ is the *deviation node* of p_k. For convenience, node s is the node deviation of p_1.

Let p_{ij} and p_{jh} be two given paths from v_i to v_j and from v_j to v_h, respectively; $p_{ij} \diamond p_{jh}$ denotes the path from v_i to v_h which is coincident with p_{ij} from v_i to v_j and with p_{jh} from v_j to v_h.

Begin {algorithm}
  Determine the shortest path from v_i to t, for any $v_i \in \mathcal{N} - \{t\}$;
  $\bar{c}_{ij} \longleftarrow \pi_j - \pi_i + c_{ij}$, for any $(v_i, v_j) \in \mathcal{A}$;
  $\mathcal{A} \longleftarrow \{a_1, \ldots, a_m\}$ where, for any $h \in \{1, \ldots, m\}$, $\bar{c}_{a_h} \leq \bar{c}_{a_{h+1}}$ if $\text{tail}(a_h) = \text{tail}(a_{h+1})$;
  $X \longleftarrow \{p_1\}$; $k \longleftarrow 0$;
  While $(X \neq \emptyset$ and $k \leq K)$ **do**
  begin
    $p \longleftarrow$ path of X such that $\bar{c}(p) \leq \bar{c}(q)$ for any $q \in X$;
    $\mathcal{P} \longleftarrow \mathcal{P} - \{p\}$;
    if (p has no cycles) **then**
    begin
      $k \longleftarrow k + 1$;
      $p_k \longleftarrow p$
    end;
    $v_i \longleftarrow$ deviation node of p;
    Repeat
      $p_{v_i} \longleftarrow$ the sub–path of p, from s to v_i;
      $a_h \longleftarrow p$ arc whose tail node is v_i;
      while (v_i is the tail node of a_{h+1}) **and** (a_{h+1} forms a cycle with p_{v_i}) **do** $h \longleftarrow h + 1$;
      if (v_i is the tail node of a_{h+1}) **then**
      begin
        $v_j \longleftarrow$ head node of a_{h+1};
        $\bar{p}_{v_j} \longleftarrow$ shortest path from v_j to t;
        $X \longleftarrow X \cup (p_{v_i} \diamond \langle v_i, a_{h+1}, v_j \rangle \diamond \bar{p}_{v_j})$
      end
      $v_i \longleftarrow$ next node in p
    until (p_{v_i} has a cycle) **or** ($v_i = t$)
  end
end. {algorithm}

We must point out that the algorithm is easily adapted for the unconstrained ranking paths problem.

3 Computational Experiments

A code for the algorithm described in this paper (**MPS** code) was compared with a code for Yen's algorithm (**YEN** code) and a code for the algorithm of Katoh, Ibaraki and Mine (**KIM** code). All the codes were written in C and the computational experiments were carried out on a DEC AXP 7610 server with a 275 MHz processor running DEC Unix 3.2. The server has 128 Mbytes of RAM and 200.9 SPECint92.

In the comparative computational experiments we used two different types of randomly generated undirected networks.

The first type concernes euclidean networks, for which the number of nodes and the number of arcs are given; each node is a point in a plane, with integer coordinates randomly generated within a predefined range. First of all and to assure connecteness of the network, an hamiltonean cycle is randomly generated; after that, each one of all the other arcs are defined by a randomly generated pair of nodes. The distance assigned to each arc is the integer value which results from its rounded euclidean distance. In this type of problems, the number of nodes assumed the values of the set $\{2000, 3000, \ldots, 9000, 10000\}$ and, for a given number of nodes, the density (ratio between the number of arcs and the number of nodes) was each one of the values $\{2, 4, 6, 8, 10\}$. For each pair (number of nodes, density), ten networks were generated, $K = 1000$ loopless paths were ranked by each one of the three codes and the average execution times computed for each loopless path. We only report (figures 1 and 2) the results for networks with 10000 nodes and densities 2 and 10, because of the similirity of all the results.

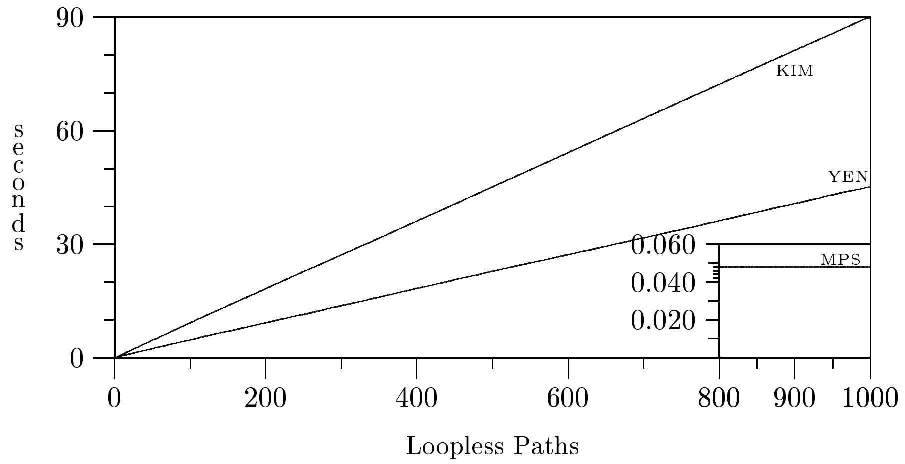

Figure 1: **Results for 10000 nodes, 20000 arcs and euclidean networks with a zooming for $K \geq 800$.**

The second type concernes grid networks. In a grid network the set of nodes is ranked by lines and by columns. A *down* $\times$ *up* grid random network is a network with *up* lines of *down* nodes each one; each node is linked with each one of its neighbors by an arc whose distance is a randomly generated integer.

In the computational experiments we used *down*$\times$25 grid random networks, with *down* in the set $\{10, 20, 25, 30, 40, 50, 100\}$. For each value of *down*, ten problems were generated, $K = 1000$ loopless paths were ranked by each one of the three codes and the average execution times computed for each loopless path. Once again the obtained results were very similar; so we only report (figure 3)

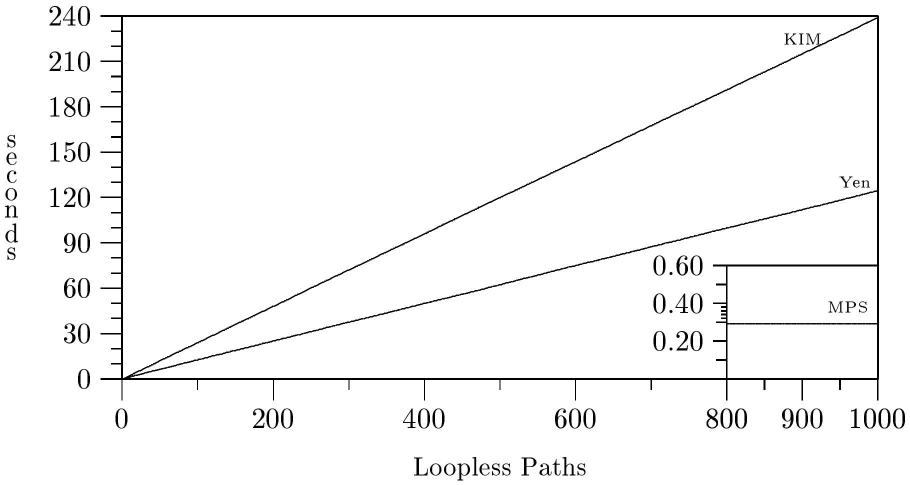

Figure 2: **Results for 10000 nodes, 100000 arcs and euclidean networks with a zooming for $K \geq 800$.**

the results for 100×25 grid networks.

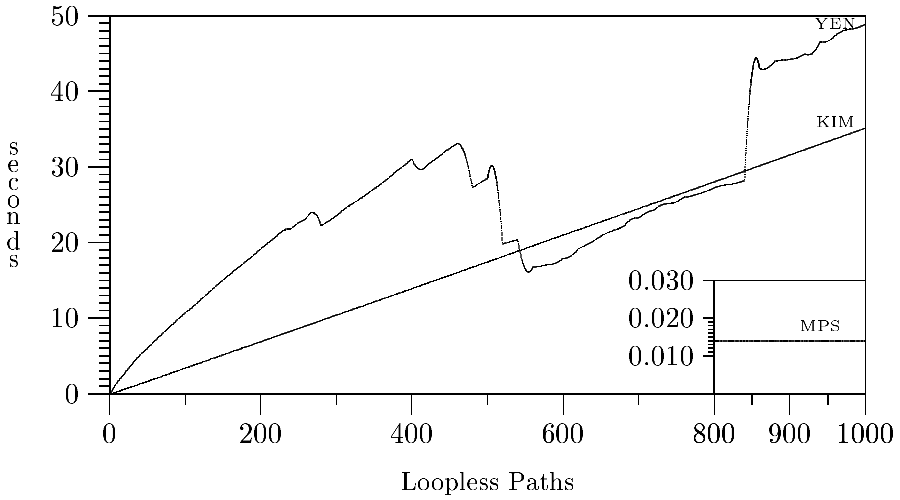

Figure 3: **Results for 100×25 grid networks with a zooming for $K \geq 800$.**

The superiority of MPS code over both KIM and YEN codes seem to be obvious (for $K = 1000$, MPS code runs from 500 to 1000 times faster than the best of the other two). Also the linearity of the results for all the codes is an immediate conclusion – the average **MPS** code times are always constant.

We must point out that the needed RAM space is not the same for all the codes; in fact, according to this parameter, KIM code is the best followed by MPS code. The reason why a peculiar shape for the average running times was obtained, is because YEN code could not even solve all the ten 100×25 grid problems until $K = 1000$.

For $K \leq 1000$, the MPS code running times are constant for each solved problem. This fact

	$K=1$		$K=100000$		**last** K				**R.T.**	
	seconds				K		seconds		seconds	
	und.	dir.	und.	dir.	und.	dir.	und.	dir.	und.	dir.
	1.019	1.098	2.035	2.127	609099	742717	2.042	2.135	0.008	0.009
	0.976	0.859	2.006	1.880	546775	837756	2.014	1.887	0.010	0.008
complete	1.000	0.991	2.034	2.017	650736	828785	2.042	2.024	0.010	0.008
1000 nodes	0.965	1.077	1.899	1.969	521609	739092	1.907	1.976	0.010	0.008
	1.016	0.941	2.042	1.976	578996	808513	2.051	1.985	0.011	0.010
average	1.008	0.978	2.003	1.994	581443.	783329.	2.011	2.001	0.010	0.009
	0.166	0.068	0.324	0.130	599193	621475	0.334	0.144	0.012	0.016
euclidean	0.149	0.066	0.299	0.130	574896	646515	0.309	0.145	0.012	0.016
10000 nodes	0.184	0.073	0.335	0.133	441301	603677	0.346	0.148	0.014	0.016
100000 arcs	0.173	0.071	0.322	0.129	526221	630502	0.334	0.146	0.014	0.017
	0.166	0.064	0.316	0.125	530606	610426	0.327	0.138	0.013	0.014
average	0.168	0.068	0.319	0.129	534443.	622519.	0.330	0.144	0.013	0.016
	2.034	2.005	d.n.r.	d.n.r.	30275	10743	2.154	2.216	0.075	0.164
	1.962	2.170	d.n.r.	d.n.r.	14441	13591	2.055	2.283	0.081	0.065
1000×25	2.122	2.183	d.n.r.	d.n.r.	16636	49314	2.238	2.316	0.068	0.084
grid	1.736	1.923	d.n.r.	d.n.r.	23579	44671	1.873	2.046	0.090	0.075
	1.878	2.225	d.n.r.	d.n.r.	31996	17557	2.001	2.329	0.077	0.055
average	1.946	2.101	——	——	23385.	27175.	2.064	2.238	0.078	0.089
	0.156	0.168	0.211	0.217	107044	210464	0.213	0.225	0.016	0.015
	0.154	0.152	d.n.r.	d.n.r.	92535	50832	0.216	0.204	0.021	0.010
150×150	0.141	0.176	d.n.r.	d.n.r.	97133	98567	0.197	0.235	0.014	0.015
grid	0.210	0.167	d.n.r.	d.n.r.	85051	80173	0.266	0.220	0.014	0.011
	0.173	0.179	d.n.r.	d.n.r.	87537	68167	0.225	0.231	0.010	0.009
average	0.167	0.168	——	——	93862.	101641.	0.223	0.223	0.015	0.012

Complete network problems used about **125 Mb** of RAM;
euclidean and grid network problems used about **105 Mb** of RAM.

und. – undirected; **dir.** – directed; **d.n.r.** – did not run
R.T. – Total time – (shortest tree time + change of variable time + arcs sort time)

Table 1: **MPS code results for large problems.**

suggested the resolution of larger problems in order to check the possibilities of MPS code. Our intent was to determine not only the number of loopless paths that could be ranked with MPS code, but also the needed CPU running time and RAM space. In these experiments we used (directed and undirected) euclidean, grid and complete networks. The results for five problems randomly generated for each instance, are presented in table 1.

We must point out that MPS code (as well as YEN code) considers all the networks as being directed since each undirected arc is equivalent to a pair of arcs with equal distance and opposite directions. According table 1 the results concerning the CPU execution time, being directed or undirected seems to be important only for euclidean networks; in fact, complete and grid networks presents quite similar execution times either for directed and undirected networks. Taking into account the number K of ranked loopless paths, problems in directed networks seems to be easier although grid networks presents no peculiar results.

An obvious conclusion follows from the network topology; in fact, grid network problems are harder to solve, since paths generated in this problems have a lot of arcs which implies more memory space and worse execution times.

Density also appears to be important when we are concerned with the execution time. For example, the average execution times for complete networks are worse than those obtained for euclidean networks, for example. However, we can not forget that density is an important parameter in the determination of the shortest tree as well as when the change of variable is executed and when the set of arcs is sorted (which is confirmed by the results in column **R.T.** of table 1).

In this paper a new algorithm for ranking loopless paths was proposed. In order to determine its theoretical complexity, a resonable upper bound for the number of paths that needs to be generated for ranking K loopless paths has to be determined which seems very difficult (or even impossible). However, the computational experiment results show the efficiency of the algorithm and its superiority over two classical algorithms, which, to the best of our knowledge, are being considered the most efficient.

Acknowledgments:

Research developed within CISUC ("Centro de Informática e Sistemas da Universidade de Coimbra") and partly supported by the PRAXIS XXI Project of JNICT ("Junta Nacional de Investigação Científica e Tecnológica").

Thanks are also due to Mark S. Amador for his help in the style.

References

[1] 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, (1994), 188–191.

[2] D. Eppstein, Finding the k Shortest Paths, 35th IEEE Symp. Foundations of Comp. Sci., Santa Fé (1994), 154–165.

[3] G. Gallo and S. Pallotino, Shortest path methods: a unifying approach, Mathematical Programming Study 26, (1986), 38–64.

[4] N. Katoh, T. Ibaraki and H. Mine, An efficient algorithm for K shortest simple paths, Networks 12, (1982), 411–427.

[5] E.L. Lawler, A Procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem, Management Science 18, (1972), 401–405.

[6] E.Q.V. Martins and J.L.E. Santos, A New Shortest Paths Ranking Algorithm, submitted.

[7] D. Shier, On algorithms for finding the K shortest paths in a network, Networks 9, (1979), 195–214.

[8] J.Y. Yen, Finding the K shortest loopless paths in a network, Management Science 17, (1971), 712–716.