Top (2007) 15: 372–382
DOI 10.1007/s11750-007-0033-0

ORIGINAL PAPER

Computational experiments with a lazy version of a K quickest simple path ranking algorithm

M. Pascoal · M.E. Captivo · J.C. Clímaco

Received: 22 June 2007 / Accepted: 22 September 2007 / Published online: 11 October 2007

Abstract The quickest path problem is related to the classical shortest path problem, but its objective function concerns the transmission time of a given amount of data throughout a path, which involves both cost and capacity. The K-quickest simple paths problem generalises the latter, by looking for a given number K of simple paths in non-decreasing order of transmission time.

Two categories of algorithms are known for ranking simple paths according to the transmission time. One is the adaptation of deviation algorithms for ranking shortest simple paths (Pascoal et al. in Comput. Oper. Res. 32(3):509–520, 2005; Rosen et al. in Comput. Oper. Res. 18(6):571–584, 1991), and another is based on ranking shortest simple paths in a sequence of networks with fixed capacity lower bounds (Chen in Inf. Process. Lett. 50:89–92, 1994), and afterwards selecting the K quickest ones.

After reviewing the quickest path and the K-quickest simple paths problems we describe a recent algorithm for ranking quickest simple paths (Pascoal et al. in Ann.

M. Pascoal (✉)
Departamento de Matemática—CIS, Faculdade de Ciências e Tecnologia, Universidade de Coimbra, Apartado 3008, 3001-454 Coimbra, Portugal
e-mail: marta@mat.uc.pt

M.E. Captivo
DEIO-CIO, Faculdade de Ciências, Universidade de Lisboa, Campo Grande, Bloco C6, Piso 4, 1749-016 Lisbon, Portugal
e-mail: mecaptivo@fc.ul.pt

J.C. Clímaco
Faculdade de Economia, Universidade de Coimbra, Av. Dias da Silva, 165, 3004-512 Coimbra, Portugal
e-mail: jclimaco@inescc.pt

J.C. Clímaco
Instituto de Engenharia de Sistemas e Computadores—Coimbra, R. Antero de Quental, 199, 3000-033 Coimbra, Portugal

Oper. Res. 147(1):5–21, 2006). This is a lazy version of Chen's algorithm, able to interchange the calculation of new simple paths and the output of each k-quickest simple path.

Finally, the described algorithm is computationally compared to its former version, as well as to deviation algorithms.

Keywords Graph algorithms · Networks · Quickest path · Ranking

Mathematics Subject Classification (2000) 90C27 · 90C35

1 Introduction

The quickest path problem (QPP) is an optimal path problem, with an objective function that represents the total transmission time of $\sigma \in \mathbb{R}^+$ data units along each path, and which is intended to be minimised. This problem was introduced in Moore (1976), and later algorithms were proposed to solve it, based on the relation with the shortest path problem (SPP) (Chen and Chin 1990; Rosen et al. 1991), or viewing it as a bicriteria problem (Martins and Santos 1997).

The generalisation of the QPP that computes $K \in \mathbb{N}$ loopless paths ordered by the total transmission time is called the quickest simple paths ranking or the K quickest simple paths problem (KQSPP). The procedures to solve this problem are either inspired by quickest path algorithms (Chen 1994), or adaptations of ranking shortest simple path algorithms (Pascoal et al. 2005; Rosen et al. 1991). The two types of methods cannot be compared, but recently a lazy version of Chen's algorithm comparable with both has been suggested (Pascoal et al. 2006) (or Pascoal et al. 2004 in an earlier version). The purpose of this work is to evaluate the performance of the variant of Chen's algorithm and to compare it with the existent methods.

The paper is organised as follows. Section 2 introduces notation and states the problem. Section 3 is devoted to a brief review of K quickest simple path algorithms, with special emphasis on the lazy version of Chen's algorithm. Finally, Sect. 4 reports and discusses the computational efficiency of the different algorithms.

2 The quickest path problem

Let $(\mathcal{N}, \mathcal{A})$ be a network with a set $\mathcal{N}$ of n nodes and a set $\mathcal{A}$ of m arcs, and let $s, t \in \mathcal{N}$ (with $s \neq t$) be the initial and terminal nodes of $(\mathcal{N}, \mathcal{A})$, respectively. The set of paths (simple paths) from s to t in $(\mathcal{N}, \mathcal{A})$ will be denoted by $\mathcal{P}$ ($\bar{\mathcal{P}}$). Two values are assigned to each arc (i, j), $c_{ij} \in \mathbb{R}_0^+$ and $u_{ij} \in \mathbb{R}^+$, which represent the cost for traversing (i, j) and its capacity, respectively. Thus, the cost and the capacity of a path p are given by $c(p) = \sum_{(i,j)\in p} c_{ij}$ and $u(p) = \min_{(i,j)\in p}\{u_{ij}\}$, and the total transmission time for sending $\sigma \in \mathbb{R}^+$ data units from s to t throughout path p is expressed by $T(p) = c(p) + \frac{\sigma}{u(p)}$. The QPP, firstly formulated by Moore (1976), is defined as $\min_{p\in\mathcal{P}}\{T(p)\}$.

Although the QPP resembles the widely-known SPP, its objective function is not additive and does not satisfy the Optimality Principle. This means that the optimal

Table 1 Worst-case complexities for quickest path algorithms

	Time	Space
Chen & Chin	$\mathcal{O}(r(m + n \log n))$	$\mathcal{O}(r(n + m))$
Rosen, Sun & Xue	$\mathcal{O}(r(m + n \log n))$	$\mathcal{O}(n + m)$
Martins & Santos	$\mathcal{O}(r(m + n \log n))$	$\mathcal{O}(n + m)$

path may contain non-optimal subpaths, and thus the quickest path cannot be found by a labelling algorithm like the ones used for finding shortest paths. Still, the first idea for solving the QPP is motivated by the facts that when the capacity is constant it coincides with the SPP, and that the number of possible path capacities is also the number of arc capacities. In fact, if $u_1, \ldots, u_r$ are the distinct arc capacities, in increasing order, and p_i denotes the shortest path such that $u(p_i) \geq u_i$, $i = 1, \ldots, r$, then the quickest path is p^* such that $T(p^*) = \min_{1 \leq i \leq r}\{T(p_i)\}$.

In Chen and Chin (1990) used this result to duplicate $(\mathcal{N}, \mathcal{A})$ into r levels, each one corresponding to a subnetwork of $(\mathcal{N}, \mathcal{A})$ where the capacity lower bound is fixed as u_i, for any $i = 1, \ldots, r$. Then, they calculated the tree of the shortest paths in the enlarged network and chose the quickest path among the shortest ones obtained. One year later Rosen et al. (1991) noted that the storage space can be reduced if considering each previous level as a subnetwork of $(\mathcal{N}, \mathcal{A})$ where the higher capacity arcs are successively deleted as new shortest paths are calculated. Like in Chen and Chin's proposal the quickest path is selected among the computed shortest paths. Martins and Santos (1997) had a different motivation and interpreted the QPP as a bicriteria problem where c is minimised and u is maximised. Their proposal is very close to the algorithm by Rosen et al., although fewer subproblems are solved because Martins and Santos determine the shortest path with maximum capacity in each subnetwork.

Table 1 summarises the theoretical complexity bounds of the mentioned quickest path algorithms.

3 *K* quickest simple path algorithms

In this paper we deal with the KQSPP, which consists of the determination of $K > 1$ simple paths in $\bar{\mathcal{P}}$ in non-decreasing order of T. In the following, p_i denotes the ith quickest simple path from s to t in $(\mathcal{N}, \mathcal{A})$.

The solution approaches for this problem can be grouped into two categories: deviation algorithms—adaptations of deviation algorithms for ranking shortest simple paths—and Chen's algorithm—adaptation of Rosen et al.'s algorithm for the QPP. Recently a lazy version of the latter algorithm was also suggested, that allows the methods in both categories to be compared. These methods, as well as the recent variant of Chen's algorithm, are now summarised.

3.1 Deviation algorithms

As mentioned above, deviation algorithms are adaptations of K shortest simple path algorithms (also known as deviation algorithms themselves) and are characterised by

using a set X of candidates for the next kth simple path. A general deviation method can be schemed as:

Algoritm 1 *Deviation algorithms to rank K quickest simple paths*
$X \leftarrow \{p_1\}$
`For` $k = 1, \ldots, K$ `Do`
  $p_k \leftarrow$ best path in X
  $X \leftarrow X - \{p_k\}$
  Generate new simple paths (p_k deviations) and store them in X
`EndFor`

The main algorithms of this type, by Yen (1971) and by Katoh et al. (1982) (this one only for undirected networks), use distinct partitions of the set $\bar{\mathcal{P}}$, which reflect on the deviation simple paths generated while analysing a given p_k.

Shortly one can say that Yen analyses every node (in the worst-case) of each p_k, which consists of solving a `SPP` in order to obtain a new deviation simple path. On the contrary, using undirected network properties Katoh et al. manage to compute the second shortest simple path by finding the trees of the shortest paths rooted at s and rooted at t (thus with the `SPP` theoretical complexity order), and therefore they are able to generate only at most 3 new deviations for each p_k.

Rosen et al. (1991) formulated the `KQSPP`, and proved that the partition introduced by Yen doesn't depend on the objective function and adapted his algorithm for ranking paths by total transmission time. Later, Pascoal et al. (2005) developed a method that uses the partition by Katoh, Ibaraki and Mine for the same problem.

3.2 Chen's algorithm

Chen (1994) looked at the problem from a different perspective. His method is analogous to the one used by Rosen et al. in the `QPP`, since it ranks the K shortest simple paths in a sequence of $(\mathcal{N}, \mathcal{A})$ subnetworks, with a fixed capacity lower bound. After these Kr simple paths have been computed and stored, the K quickest simple paths are selected by order among them. Any shortest simple path ranking algorithm can be applied for this purpose and this may modify the theoretical and computational behaviour. It should be noted that a path may belong to several networks; therefore, it can be computed more than once.

As mentioned before, this is a rather different approach from the ones described above since the computation is separated into the calculation phase and the K quickest simple paths output. This type of approach can be useful if the algorithm application allows the computational paths to be stored, and then output as needed. Some disadvantages can also be pointed out, like:

- A high number of simple paths has to be determined, always Kr.
- Still as a consequence of the previous point, the computation phase is usually long; therefore, the output of the quickest path can take a long time.
- K has to be known in advance.

In terms of the execution time, this results in a sharp increase in the phase of determining the Kr simple paths, and it becomes almost irrelevant when selecting the

K best ones. Thus, the comparison of Chen's algorithm with deviation algorithms is difficult, since the latter alternate the determination of each p_k and the computation of new candidates.

3.3 A variant of Chen's algorithm

A new variant of Chen's algorithm was recently suggested by Pascoal et al. (2006) (or Pascoal et al. 2004). The motivation for developing this variant was to design a Chen-like method, based on ranking simple paths by order of cost in a sequence of networks, able to alternate paths computation and paths output, and thus closer to deviation algorithms. The resulting method is named "lazy version" of the former algorithm, in the sense that it should generate new simple paths only as they are needed.

Assume p_k has been determined by the algorithm. Then p_{k+1} is the shortest simple path P_i in $(\mathcal{N}, \mathcal{A}(u_i))$, $i = 1, \ldots, r$, with the least total transmission time, which has not been selected as $p_1, \ldots, p_k$. The selected simple path, for instance P_j, is replaced by another simple path in $(\mathcal{N}, \mathcal{A}(u_j))$, which should be the one following P_j in $(\mathcal{N}, \mathcal{A}(u_j))$ in terms of cost. The variant of Chen's algorithm is summarised below.

Algoritm 2 *Variant of Chen's algorithm to rank K quickest simple paths*

$(u_1, \ldots, u_r) \leftarrow$ arc capacity values in increasing order
$\mathcal{A}' \leftarrow \mathcal{A}$
`For` $(i \in \{1, \ldots, r\})$ `Do`
  $\mathcal{A}' \leftarrow \{(x, y) \in \mathcal{A}' : u_{xy} \geq u_i\}$
  $P_i \leftarrow$ the shortest path in $(\mathcal{N}, \mathcal{A}')$
`EndFor`
$k \leftarrow 0$
`While` $(k < K$ and $\{P_1, \ldots, P_r\} \neq \emptyset)$ `Do`
  $p \leftarrow P_i$ such that $T(P_i) = \min\{T(P_j) : 1 \leq j \leq r\}$
  $\mathcal{A}' \leftarrow \{(x, y) \in \mathcal{A} : u_{xy} \geq u_i\}$
  $P_i \leftarrow$ the next shortest simple path in $(\mathcal{N}, \mathcal{A}')$
  `If` $(p \notin \{p_1, \ldots, p_k\})$ `Then`
    $k \leftarrow k + 1$
    $p_k \leftarrow p$
  `EndIf`
`EndWhile`

4 Algorithms comparison

4.1 Theoretical comparison

The computational order of Chen's algorithm depends on the method used for ranking shortest simple paths, since it solves that problem in r networks. As Yen's algorithm has worst-case complexity order of $\mathcal{O}(Kn(m + n \log n))$ and Katoh et al.'s of

Table 2 Worst-case complexities for K-quickest simple path algorithms

	Time	Space
Rosen, Sun & Xue	$\mathcal{O}(Knr(m+n\log n))$	$\mathcal{O}(Kn+m)$
Pascoal, Captivo & Clímaco	$\mathcal{O}(Kr(m+n\log n))$	$\mathcal{O}(Kn+m)$
Chen (with Yen)	$\mathcal{O}(Knr(m+n\log n))$	$\mathcal{O}(Krn+m)$
Chen (with Katoh et al.)	$\mathcal{O}(Kr(m+n\log n))$	$\mathcal{O}(Krn+m)$

$\mathcal{O}(K(m+n\log n))$, then Chen's algorithm may be of $\mathcal{O}(Krn(m+n\log n))$ if using the first one as a subprocedure, or of $\mathcal{O}(Kr(m+n\log n))$ if using the second one. These time complexity bounds are analogous for Rosen et al.'s and for Pascoal et al.'s methods, respectively. The differences in the two types of algorithms become evident in the memory space complexity bound of $\mathcal{O}(m+Krn)$ for the original version of Chen's algorithm, since this method always demands that Kr simple paths are generated. As for deviation algorithms, only K simple paths need to be computed; therefore, its worst-case space order is $\mathcal{O}(m+Kn)$. Table 2 summarises these theoretical complexity bounds.

In general, the described variant for the `KQSPP` should determine fewer simple paths than the original Chen's algorithm. In fact, optimistically the number of simple paths generated is K, so the best-case theoretical space complexity is $\Omega(Kn+m)$. However, the number of simple paths obtained by Chen's algorithm upper bounds its variant, so the worst-case complexity in terms of memory space is still $\mathcal{O}(Krn+m)$. In an average case it is expected that this bound is rarely achieved, as the empirical experiments in the next section show. Because the theoretical worst-case for this variant coincides with Chen's original algorithm its time complexity is $\mathcal{O}(Knr(m+n\log n))$ or $\mathcal{O}(Kr(m+n\log n))$, when using Yen's or Katoh et al.'s algorithm, respectively, as the original algorithm.

It seems clear that the new variant outperforms the former version of Chen's algorithm. It is more difficult, though, to predict what its behaviour will be in comparison with deviation algorithms. In fact, the first one maintains r shortest simple path rankings, while the second ranks quickest simple paths using a partition for ranking shortest simple paths. Hence, on the one hand this second approach only has to consider one ranking (easier than using r), but on the other hand, the subproblems it has to solve are quickest path problems (harder than shortest path problems).

The initialisation phase of the first method consists in computing and storing r shortest paths, while with the second r paths are computed, at most, and only 1 is stored. After the initialisation the first method selects the best path among $P_1,\ldots,P_r$ and replaces it by the next shortest simple path, which seems easier than choosing the best among all candidate simple paths generated so far and replacing it by the next quickest path (after calculating at most r shortest paths). This phase seems more simple for the variant of Chen's algorithm, but it should be noted that repeated paths may be obtained; therefore, the cycle can be repeated more than K times. Besides, it is harder to manipulate the data structure that maintains several rankings at the same time.

Table 3 Mean values for random networks, $n = 1000$

	$d = 2$			$d = 10$		
	CA	RA	NA	CA	RA	NA
Computed deviations	23036	1057	8470	47901	470	14412
Heap/p_1 time (s)	3.712	0.001	0.105	22.596	0.004	0.345
Total time (s)	3.718	0.328	1.460	22.627	1.394	7.412

4.2 Empirical experiments

For an empirical evaluation of the recent algorithm in comparison with the older ones, several tests were performed on connected networks. Two different sets of test problems were used. For both sets the costs and the capacities were uniformly generated in $\{1, \ldots, 100\}$, but in the second the number of distinct capacities was previously set to $r = 5, 10, \ldots, 25, 30$. Moreover, Set 1 was formed by the following network classes:

- Random, with $n = 1000, 5000$ and average node degree $d = m/n = 2, 10$
- Complete, with $n = 100, 200, 300$
- Grid, with $p \times 50$ and $p = 50, 100, 150, 200$

while Set 2 consisted of random networks with $n = 1000, 5000$ and $d = 2, 10$. In every problem $\sigma = 100$ data units to transmit from s to t were considered and $K = 100$ simple paths were ranked between those two nodes. The following results are average values obtained in 10 networks of each type and size.

The three types of algorithm were expected to perform identically either when based on Yen's procedure or when based on Katoh et al.'s procedure. For that reason, and also for simplicity, we chose to code only Yen's versions of those algorithms, namely: the original Chen's algorithm (CA), the deviation algorithm by Rosen et al. (RA), and the lazy version of Chen's algorithm (NA). The codes were written in C language and the tests were performed on a Pentium 4 at 2.4 GHz, with 1 GB of RAM, running Linux.

In the study that follows we analyse several aspects of the CPU times needed by the coded algorithms. First, we consider the time demanded until the algorithm can output simple path p_1, which in CA corresponds to the time for constructing a heap with the Kr computed simple paths, while for the remaining algorithms it represents the time for computing p_1. Second, the execution time until p_K is known. With the aim of comparing RA and NA the partial times are also analysed, that is, the time to output p_i after p_{i-1} has been listed, for any $i = 2, \ldots, K = 100$. In CA this corresponds only to the time to select p_i after p_{i-1}, while in the other two codes it is the time to select and analyse p_i.

Tables 3 and 4 present some of these values for the tests on random networks: the number of computed simple paths, phase 1 time, and the total time (both in seconds). As it has been said, the differences between Chen's algorithm and deviation algorithms are obvious in the time for phase 1, where RA clearly outperforms CA. The lazy variant takes advantage of the simplified phase 1 and the CPU times show a big

Table 4 Mean values for random networks, $n = 5000$

	$d = 2$			$d = 10$		
	CA	RA	NA	CA	RA	NA
Computed deviations	28310	1143	11268	45810	498	10665
Heap/p_1 time (s)	22.225	0.005	0.112	144.947	0.033	0.648
Total time (s)	22.236	2.501	9.468	144.978	11.529	34.717

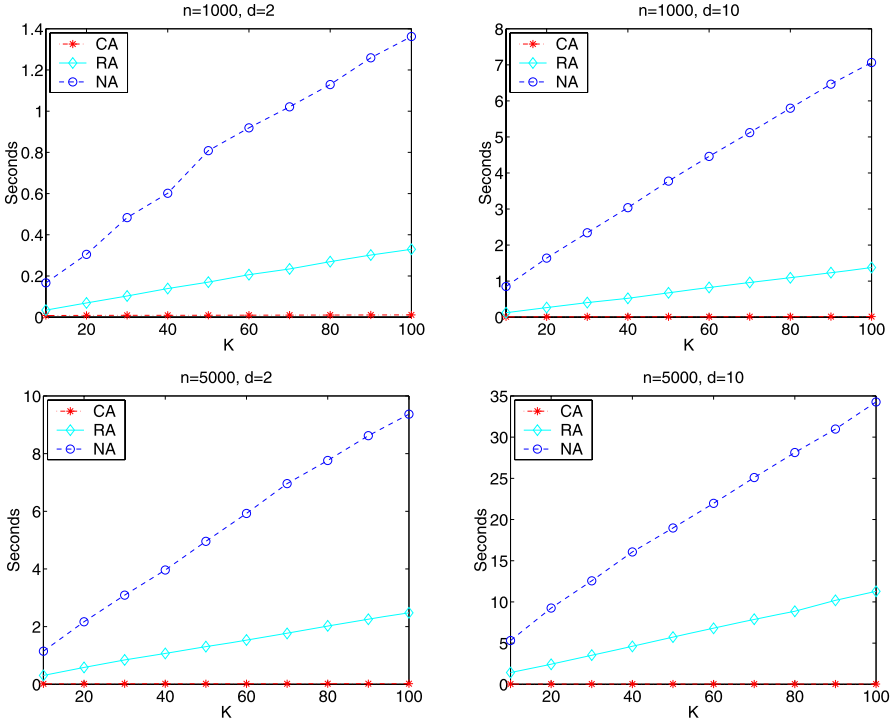

Fig. 1 Partial times on random networks, Set 1

improvement when comparing it to CA, although these results are still worse than the ones presented by RA. In terms of the total CPU times the results are analogous, CA being the worst code, followed by NA, and only then by RA.

Figure 1 presents the partial times of the 3 coded algorithms in random networks of Set 1, while Figs. 2 and 3 only show some results for RA and NA in the remaining networks of the same set, as these values are worthless for CA.

In the remaining networks of Set 1 these results appear to indicate that the partial times increase linearly with the number K of listed simple paths and that the lazy implementation of Chen's algorithm behaves worse than deviation algorithms. The partial times for NA grow faster than RA in Figs. 1 and 2, and the difference among these values on grid networks is even more obvious—see Fig. 3. In fact, the behaviour

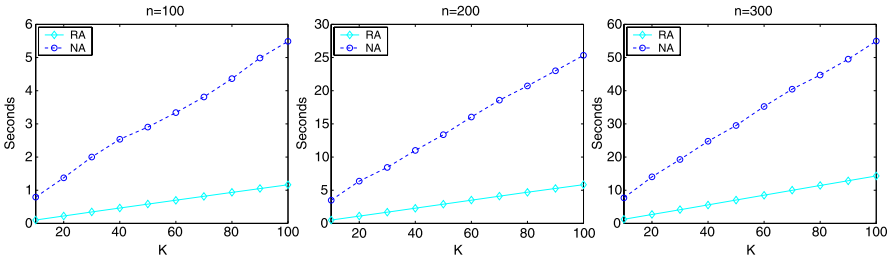

Fig. 2 Partial times on complete networks, Set 1

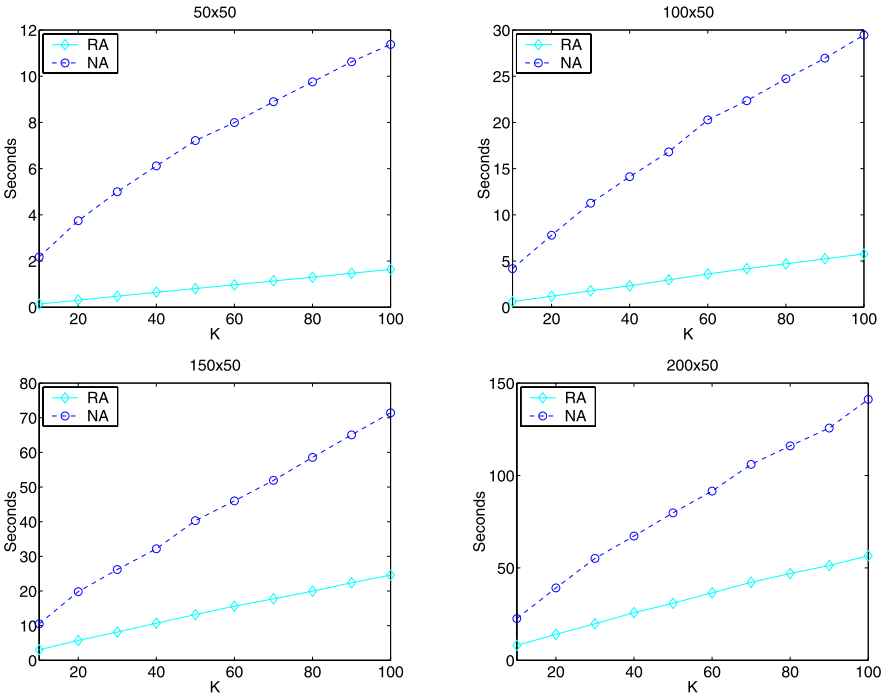

Fig. 3 Partial times on grid networks, Set 1

of both codes depends on the size of the network (in terms of n as well as for m) but NA was more sensitive to the variation of those factors.

Based on the results presented for Set 1 it seems that the proposed version of Chen's algorithm indeed improves the former version, although deviation algorithms are still those with the best performance. As said in the previous section this might be due to the difficulty in maintaining r simple path rankings at the same time. This suggested to run a second set of experiments where the number r of capacities is fixed a priori as a small value, in order to evaluate the weight of r on the lazy algorithm's performance.

Table 5 Mean total CPU time (seconds) for random networks, $n = 1000$

	$d = 2$			$d = 10$		
	CA	RA	NA	CA	RA	NA
$r = 10$	0.512	0.213	0.233	2.604	1.081	0.648
$r = 20$	0.802	0.253	0.443	5.020	1.223	1.344
$r = 30$	1.174	0.300	0.454	7.622	1.340	2.129

Table 6 Mean total CPU time (seconds) for random networks, $n = 5000$

	$d = 2$			$d = 10$		
	CA	RA	NA	CA	RA	NA
$r = 10$	4.053	1.397	0.951	18.367	9.074	3.708
$r = 20$	5.054	1.758	1.970	34.375	12.139	7.748
$r = 30$	13.137	2.159	2.682	48.635	12.327	9.962

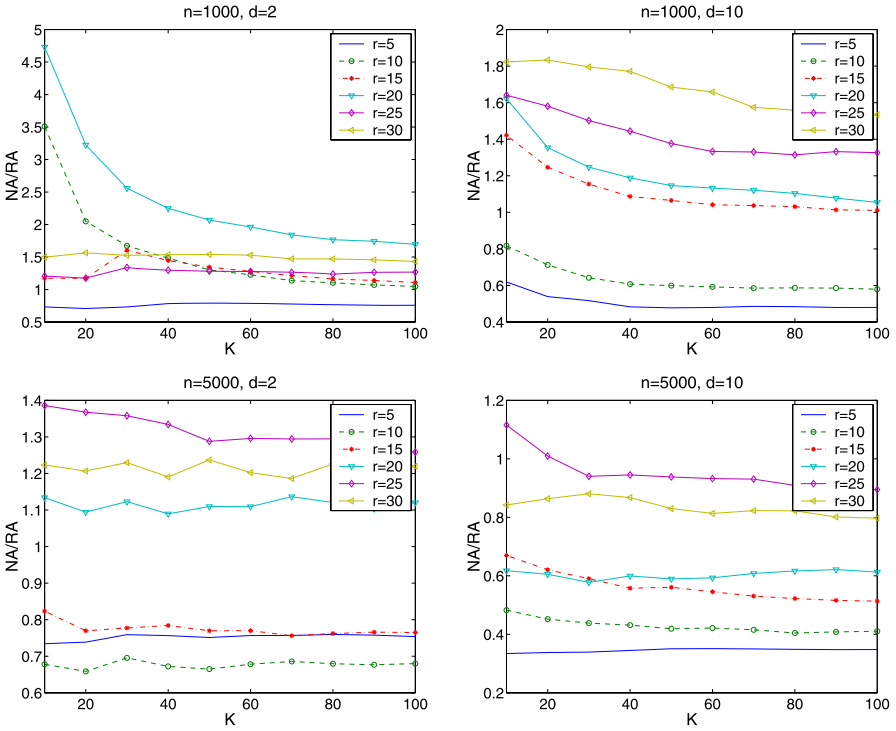

Fig. 4 Partial times on random networks, Set 2

Tables 5 and 6 summarise the total CPU times for the three coded algorithms on this second set of networks. Once again CA was the code with the worst results, but

now the total times of RA and NA are closer, and NA even performed better than RA for networks where r was small and also when the networks were bigger.

Figure 4 shows the quotient $\frac{\text{NA partial times}}{\text{RA partial times}}$ for the networks in Set 2, which should be greater than 1 when RA outperforms NA. The results are not always consistent but it seems that the conclusions drawn for the total times are still valid for the partial ones, that is, NA is better than RA when there are few distinct capacities and the density of the network increases.

5 Conclusions

A recent algorithm for ranking K quickest simple paths was discussed, and computational experience on several types of networks comparing it with the former version and with deviation algorithms was presented.

The lazy version of Chen's algorithm obtained CPU times that grow linearly with K and increase with n, depending also on the type of network. This new procedure behaved better than the former version of Chen's algorithm. However, it still spends a lot of time in the initialisation phase, to determine p_1 and to keep r simple paths rankings at a time. Moreover, for high values of r deviation algorithms outperformed the lazy variant, but for small r's the lazy algorithm was more efficient.

References

Chen YL (1994) Finding the K quickest simple paths in a network. Inf Process Lett 50:89–92

Chen YL, Chin YH (1990) The quickest path problem. Comput Oper Res 17(2):153–161

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

Martins EQV, Santos JLE (1997) An algorithm for the quickest path problem. Oper Res Lett 20:195–198

Moore MH (1976) On the fastest route for convoy-type traffic in flowrate-constrained networks. Trans Sci 10:113–124

Pascoal MMB, Captivo MEV, Clímaco JCN (2004) On the quickest path problem. Working paper n 15, CIO (http://cio.fc.ul.pt/files/15.2004.pdf)

Pascoal MMB, Captivo MEV, Clímaco JCN (2005) An algorithm for ranking quickest simple paths. Comput Oper Res 32(3):509–520

Pascoal MMB, Captivo MEV, Clímaco JCN (2006) A comprehensive survey on the quickest path problem. Ann Oper Res 147(1):5–21. Special issue on Multiple Objective Combinatorial and Discrete Optimization

Rosen JB, Sun SZ, Xue GL (1991) Algorithms for the quickest path problem and the enumeration of quickest paths. Comput Oper Res 18(6):571–584

Yen JY (1971) Finding the K shortest loopless paths in a network. Manag Sci 17:712–716