

The minmax regret robust shortest path problem in a finite multi-scenario model

Marta M. B. Pascoal*, Marisa Resende

March 31, 2014

Department of Mathematics, University of Coimbra
Apartado 3008, EC Santa Cruz, 3001-501 Coimbra, Portugal
Phone: +351 239 791150, Fax: +351 239 832568

Institute for Systems Engineering and Computers – Coimbra (INESCC)
Rua Antero de Quental, 199, 3000-033 Coimbra, Portugal

E-mails: {marta, mares}@mat.uc.pt

Abstract

The robust shortest path problem is a network optimization problem that can be defined to deal with uncertainty of costs associated with the arcs of a network. Two models have been considered for the robust shortest path problem: interval data and discrete data sets. This work addresses the robust shortest path problem with a minmax regret objective function on a finite multi-scenario model. This problem consists in finding an optimal path in the sense that it has the minimum maximum deviation from the shortest one over all scenarios. With this goal some properties of the problem and of its optimal solutions are derived. These results allow to introduce three approaches, a labeling algorithm, an algorithm based on ranking loopless paths, and a hybrid algorithm which ranks loopless paths in a suitable way to apply the early elimination of useless solutions. The algorithms are tested on random networks and compared with a previous method for the same problem. The obtained computational results are reported and discussed. They show that the labeling and the hybrid approaches outperform the others.

Keywords: network, scenario, minmax regret, robust shortest path, labeling, ranking

1 Introduction

Finding a shortest path between two nodes of a network is a well known problem that consists in computing a path with the least cost, when each arc is associated with a single deterministic value. Trying to model reality, these cost values are not always known or sometimes they are not accurate. In [6], Dias and Clímaco assume that only partial information is known about the problem, address the shortest path problem from a multicriteria and a decision making point of view and present algorithms to compute the set of non dominated paths taking into account the available information, under certain conditions. Perny and Spanjaard [20] make a similar assumption and deal with dominance rules in a state space graph for the same purpose. Another approach that has been used to cope with uncertainty is robust optimization [12]. Two types of uncertainty models have been considered: interval data models and discrete data models. In the first of these models the costs range within given intervals, whereas in the second the costs can belong to discrete finite sets. When the information given in different scenarios is aggregated, several criteria can be used. One of the most common

*Corresponding author

aims at minimizing the worst case for all scenarios. When the goal is to find paths between a pair of nodes, two versions can be considered. The first, known as minmax shortest path problem or absolute robust shortest path problem, consists of finding the path with the minimum maximum cost over all scenarios. The second, known as minmax regret robust shortest path problem or robust deviation shortest path problem, aims at finding the path with the minimum maximum deviation cost with respect to the optimum solution in each scenario. The present work is dedicated to the latter problem and considers a finite set of possible cost scenarios.

In 1992, the absolute version of the robust shortest path problem with a finite set of scenarios was studied by Murthy and Her [18]. They introduced a labeling algorithm based on a multicriteria approach to solve the problem exactly. The method includes a dominance test between labels as well as pruning techniques developed with the goal of discarding some of them. Later, in 1998, Yu and Yang [22] introduced the relative version of the problem with a discrete set of scenarios. This work presented pseudo-polynomial algorithms for the problem, based on dynamic programming, as well as a more specific method designed for layered networks. It was also shown that the problem is strongly NP-hard if the number of scenarios is unlimited. Then, a heuristic was developed to compute an approximate optimal solution, since the exact approach revealed to be ineffective for problems with large cost upper bounds or a large number of scenarios. In 2010, Bruni and Guerriero [2] developed heuristic rules for guiding the search performed by Murthy and Her's algorithm for the absolute robust shortest path problem.

Other works have focused on a similar problem but considered that each arc cost ranges within a given interval. The first work on this subject was proposed in 2001, by Karasan, Pinar and Yaman, and addressed the case of acyclic networks [11]. In this paper a finite number of scenarios containing the optimum solution could be determined by combining the upper and lower limits of the cost intervals. Several exact techniques were introduced later by Montemanni et al., which extended the determination of a robust shortest path for general networks [15, 16, 17]. More recently, Catanzaro, Labbé and Salazar-Neumann [4] proposed preprocessing tools for reducing a network with the goal of speeding up the determination of a robust shortest path on an interval data model. Recent surveys on these topics can be found in [3, 9, 10].

The present work considers the minmax regret as the robustness criterion on a finite multi-scenario model to determine an optimal path between a pair of nodes of a network. Three exact methods are developed for dealing with the robust shortest path problem. The first is a labeling algorithm including cost lower and upper bounds to discard uninteresting solutions. The second is an algorithm supported by a loopless paths ranking under a specific scenario and imposing halting conditions defined by the costs of the solutions determined along the process. Finally, the third is also a ranking approach enhanced with cost bounds, similar to the first method, designed with the purpose of fostering the generation of new paths and of cost bounds that will likely lead to a smaller number of iterations than the second method.

The next section is dedicated to the definition of the minmax regret robust shortest path problem, the introduction of notation and the derivation of properties concerning the optimum solution and its cost. Section 3 is devoted to the algorithms presentation. In Section 4 an example of the application of the three algorithms is presented. In Section 5 computational experiments comparing the proposed algorithms and the method by Yu and Yang are presented and the obtained results are discussed. Conclusions and comments on future research are drawn in the last section.

2 Problem definition and notation

Hereinafter a finite multi-scenario model is represented as $G(V, A, S_k)$, where G is a directed graph with a set of nodes $V = \{1, \dots, n\}$, a set of m arcs $A \subseteq \{(i, j) : i, j \in V \text{ and } i \neq j\}$ and a finite set of scenarios $S_k := \{1, \dots, k\}$, $k > 1$. For each arc $(i, j) \in A$, i and j are named the tail and the head node, respectively, and c_{ij}^s represents its cost in scenario s , $s \in S_k$. It is assumed that the graph contains no parallel arcs.

A path from i to j , $i, j \in V$, in graph G , also called an (i, j) -path, is an alternating sequence of nodes and arcs of the form

$$p = \langle v_1, (v_1, v_2), v_2, \dots, (v_{r-1}, v_r), v_r \rangle,$$

with $v_1 = i$, $v_r = j$ and where $v_l \in V$, for $l = 2, \dots, r-1$, and $(v_l, v_{l+1}) \in A$, for $l = 1, \dots, r-1$. The sets of arcs and of nodes in a path p are denoted by $A(p)$ and $V(p)$, respectively. Given two paths p, q , such that the destination node of p is also the initial node of q , the concatenation of p and q is the path formed by p followed by q , and is denoted by $p \diamond q$.

Because it is assumed that graphs do not contain parallel arcs, in the following paths will be represented simply by their sequence of nodes. A cycle, or loop, is a path from a node to itself. A path is said to be loopless if all its nodes are different.

The cost of a path p in scenario s , $s \in S_k$, is defined by

$$c^s(p) = \sum_{(i,j) \in A(p)} c_{ij}^s. \quad (1)$$

With no loss of generality, 1 and n denote the origin and the destination nodes of the graph G , respectively. For simplicity of presentation, it will also be assumed that no arcs arrive at 1 and no arcs start at n in G . The set of all $(1, n)$ -paths in G is represented by P .

Let $p_{ij}^{l,s}$ represent the l -th shortest (i, j) -path in G , $i, j \in V$, for a given scenario $s \in S_k$. In order to simplify the notation, $p^{l,s}$ is used to denote the l -th shortest $(1, n)$ -path in scenario s , i.e. $p_{1n}^{l,s}$, and LB_i^s is used to denote the cost of the shortest (i, n) -path in scenario s , i.e. $c^s(p_{in}^{1,s})$.

The minmax regret robust shortest path problem corresponds to determining a path in P with the least maximum robust deviation, i.e. satisfying

$$\arg \min_{p \in P} RC(p), \quad (2)$$

where $RC(p)$ is the robustness cost of p , defined by

$$RC(p) := \max_{s \in S_k} RD^s(p), \quad (3)$$

where $RD^s(p)$ represents the robust deviation of a path p under scenario s , $s \in S_k$, defined by

$$RD^s(p) := c^s(p) - LB_1^s. \quad (4)$$

Any optimal solution of (2) is called a robust shortest path.

The set of scenarios in which $RC(p)$ occurs, i.e. the set of scenarios for which the robust deviation of $p \in P$ is maximized, will be denoted by $S(p) := \{\arg \max_{s \in S_k} RD^s(p)\}$.

The idea behind minimizing the maximum robust deviation is to find a path with the best deviation in all scenarios, with respect to the shortest path in each one. A problem that resembles this one is the minmax shortest path problem [18]. The latter is an absolute version of problem (2), for which the robust

deviation of a path p is replaced by its cost in each scenario. That is, the objective function to minimize is $\max_{s \in S_k} \{c^s(p)\}$. Both problems have the same optimum solution if the shortest paths over all scenarios have the same cost in the scenario where their cost is minimum, that is LB_1^s is a constant for any $s \in S_k$.

2.1 Properties of the optimal solutions

A robust shortest path may not be unique, as shown by the network depicted in Figure 1. In this example, a case with two scenarios, the paths $p^{1,1} = \langle 1, 2, 4 \rangle$ and $p^{1,2} = \langle 1, 3, 4 \rangle$ are the shortest from 1 to 4 for scenarios 1 and 2, respectively. There exist two $(1, 4)$ -paths, $p^{1,1}$ and $q = \langle 1, 2, 3, 4 \rangle$, with the minimum robustness cost 2 ($S(p^{1,1}) = \{2\}$ and $S(q) = \{1\}$). Therefore, they are both robust shortest paths.

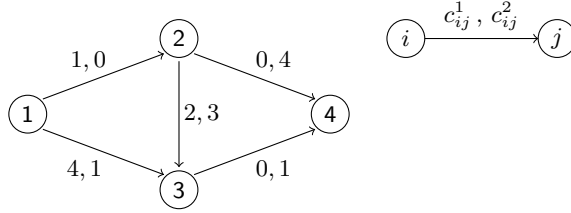


Figure 1: Network

The following result is a consequence of definitions (3) and (4).

Lemma 1. *For every $p \in P$, $RC(p) \geq 0$. Moreover, $RC(p) = 0$ if and only if p is a shortest path in every scenario $s \in S_k$.*

Taking (2) and Lemma 1 into account, one can also establish under what conditions a shortest path for a scenario can be a robust shortest path as well.

Lemma 2. *If $p \in P$ is a shortest path in every scenario $s \in S_k$, then p is a robust shortest path and $S(p) = S_k$.*

In order to develop algorithms that compute a path with the minimum robustness cost, other properties must be established. An important result concerns the cyclic nature of an optimal solution. In fact, any robust shortest path on an acyclic network G is naturally loopless. Nevertheless, when G is a general network, there may exist robust shortest paths that contain loops, as shown in Figure 2, resultant from the inclusion of arc $(2, 1)$ in the network of Figure 1. Such inclusion does not affect the optimality of the loopless paths $p^{1,1} = \langle 1, 2, 4 \rangle$ and $q = \langle 1, 2, 3, 4 \rangle$, however, $q' = \langle 1, 2, 1, 2, 4 \rangle$ is a new robust shortest $(1, 4)$ -path containing the loop $\langle 1, 2, 1 \rangle$, given that $RC(q') = RC(p^{1,1}) = RC(q) = 2$ ($S(q') = S_2$).

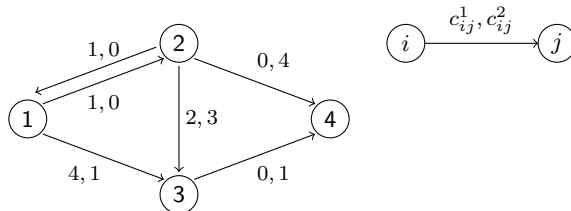


Figure 2: Network with a cycle

Although a robust shortest path may not be unique, Yu and Yang [22] proved the existence of a loopless one considering networks with non-negative arc costs. This result is still valid for networks without cycles with negative cost in any scenario, as stated in Lemma 3.

Lemma 3. *Let G be a network with no cycles with negative cost in any scenario, then there exists a loopless optimal solution of (2) in G .*

Proposition 1 presents another property of robust shortest paths that will be used later, concerning an upper bound on its cost under particular conditions.

Proposition 1. *Let $p \in P$. If q is any robust shortest path, then $c^s(q) \leq c^s(p)$, for any scenario $s \in S(p)$.*

Proof. Let $p \in P$ and s be any element of $S(p)$. Attending to the definition of $S(p)$ and to (3), $RC(p) = RD^s(p)$. By contradiction, assume q is a path satisfying $c^s(q) > c^s(p)$. Then, attending to (3) and (4), one deduces that $RC(q) \geq RD^s(q) > RD^s(p) = RC(p)$. Consequently, q cannot satisfy (2) and be a robust shortest path. \square

3 Algorithms

The results established at Subsection 2.1 are a motivation for developing three algorithms to solve (2). All of them allow to obtain a loopless robust shortest $(1, n)$ -path. The first one is based on a labeling approach, whereas the others are based on the ranking of loopless paths by non-decreasing order of their costs for a fixed scenario and use the cost upper bound introduced in Proposition 1. Moreover, the third is a hybrid version of the previous two, which uses both ranking and pruning techniques. The methods are presented in the following.

3.1 Labeling algorithm

The first method presented here for computing a robust shortest $(1, n)$ -path is a labeling approach inspired on the algorithm proposed by Murthy and Her [18] for the minmax shortest path problem. This problem does not satisfy Bellman's principle of optimality, that is to say that an optimal path may contain sub-paths that are not optimal. Therefore, Murthy and Her consider each scenario as one criterion and develop a labeling algorithm combined with dominance tests between labels to solve the problem. The method is complemented by rules for pruning unnecessary labels. One of them is based on the use of lower bounds with respect to each cost function of a path from a node to n . The other results from the Lagrangian relaxation of the subproblem of the linear programming formulation obtained when the previous bounds are fixed. Even though the robust shortest path problem cannot be solved by exactly the same process, the algorithm described in the following has a similar inspiration. The main modification is the adaptation of the upper bounds for the values of the current objective functions. First, some concepts and notation are introduced.

Let $z_i = (z_i^1, \dots, z_i^k)$ denote a label associated with a $(1, i)$ -path, p_{1i} , or with node $i \in V$. More than one $(1, i)$ -path can be eligible to become part of the solution, and thus more than one label can be associated with node i . In general multicriteria shortest path problems the labels z_i represent the cost vector of the associate $(1, i)$ -path. Here they play a similar role. However, because the objective function to evaluate $(1, n)$ -paths depends on the robustness deviation with respect to the shortest path in all scenarios, in order to simplify intermediate calculations, the label of node 1 is given by

$$z_1 = (-LB_1^1, \dots, -LB_1^k).$$

Each component z_i^s of a label z_i , $i \in V$, is related with the cost of the associate $(1, i)$ -path in scenario $s \in S_k$. Moreover, along the algorithm, when such label is selected, new labels z_j can be created for any $(i, j) \in A$, according to the formula

$$z_j = (z_i^1 + c_{ij}^1, \dots, z_i^k + c_{ij}^k). \quad (5)$$

With the above initialization, $z_n = (RD^1(p_{1n}), \dots, RD^k(p_{1n}))$ is the vector of robust deviations of a given $(1, n)$ -path, p_{1n} . Hence, the solution space is explored through the labels for node n and the optimum value is obtained by selecting the label with the least maximum component, that is, the least robustness cost of the $(1, n)$ -paths. This result is stated in the following lemma.

Lemma 4. *Let z_n be a label for node n . The $(1, n)$ -path associated with z_n is a robust shortest path if and only if*

$$\max_{s \in S_k} \{z_n^s\} \leq \max_{s \in S_k} \{z_n'^s\} \quad (6)$$

for any label z_n' associated with the destination node n .

Lemma 4 allows to eliminate labels z_n that do not satisfy (6). Nevertheless, extending the $(1, i)$ -paths associated with all the existent labels z_i , $i \in V \setminus \{n\}$, to all possible paths with destination node n can be computationally demanding. Therefore, two pruning techniques will be derived with the aim of discarding in an early stage of the algorithm some of such labels that cannot be part of an optimal loopless solution, which allows to reduce the total number of node labels that have to be stored along the calculations. For the first reduction rule, new concepts related with the dominance of the generated labels are given.

Definition 1. *Let z_i' and z_i'' be two labels associated with node i . Then z_i' dominates z_i'' if*

$$z_i'^1 \leq z_i''^1, \dots, z_i'^k \leq z_i''^k$$

and at least one of the inequalities is strict.

Definition 2. *A label associated with node i is efficient (or non dominated) if there is no other label for node i that dominates it.*

Checking the dominance between different labels of a node is crucial to decide which can be discarded. In fact, any label z_i , $i \in V \setminus \{n\}$, dominated by another label z_i' can be eliminated. Proposition 2 shows that any extension of the $(1, i)$ -path associated with z_i produces a $(1, n)$ -path with a robustness cost that is never smaller than the robustness cost of the $(1, n)$ -path resultant by the same extension from the $(1, i)$ -path associated with z_i' . The storage of repeated labels of a node can also be avoided, since robust shortest $(1, n)$ -paths containing their associate partial paths have the same robustness cost.

Proposition 2. *Let z_i and z_i' be two different labels for a node $i \in V \setminus \{n\}$, such that z_i dominates z_i' . Then, extending the $(1, i)$ -paths associated with z_i and z_i' by the same (i, n) -path, results on $(1, n)$ -paths with labels z_n and z_n' , respectively, satisfying*

$$\max_{s \in S_k} \{z_n^s\} \leq \max_{s \in S_k} \{z_n'^s\}.$$

Proof. Let p_{in} denote the (i, n) -path that extends the $(1, i)$ -paths associated with labels z_i and z_i' into $(1, n)$ -paths with labels z_n and z_n' , respectively.

By assumption, z_i dominates z_i' . Therefore, $z_n^s = z_i^s + c^s(p_{in}) \leq z_i'^s + c^s(p_{in}) = z_n'^s$, for every $s \in S_k$, and $z_n^l < z_n'^l$, for some $l \in S_k$. Consequently, $\max_{s \in S_k} \{z_n^s\} \leq \max_{s \in S_k} \{z_n'^s\}$. \square

A similar reasoning can show that two labels with equal components, for the same node and associated with different partial paths, lead to $(1, n)$ -paths with the same robustness cost when extended with the same path. Labels under those conditions will be called equivalent labels. As mentioned earlier, it is intended to restrict the search to loopless $(1, n)$ -paths. These two facts lead to an implementation that discards equivalent labels of a node and manages the labels following a first in first out (FIFO) policy. This means that breadth-search is used to build the search tree and that, when equivalent labels occur, only the first is stored. Proposition 3 shows that there is a robust shortest loopless $(1, n)$ -path under these conditions.

Proposition 3. *Assume that the set of node labels is managed as a FIFO list. Then there is a robust shortest loopless $(1, n)$ -path the sub-paths of which are associated with the first label of each node, when it has several equivalent labels.*

Proof. By contradiction, assume that no optimal loopless path exists with all nodes associated with the earliest possible label, when there are equivalent labels. Let p^* be a robust shortest loopless $(1, n)$ -path and $j \in V(p^*)$ be its node closest to n for which there are several equivalent labels. Let z_j^* be the label associated with the $(1, j)$ -path in p^* , p_{1j}^* , and suppose z_j' is the first label created for node j that is equivalent to z_j^* . Assume z_j' is associated with the $(1, j)$ -path p_{1j}' . Denote by p_{jn}^* the (j, n) -path in p^* . Then $p_{1j}' \diamond p_{jn}^*$ is a $(1, n)$ -path, with the same robustness cost as p^* . Moreover, it is associated with z_j' , the first label of j , therefore, by assumption it should contain a loop. Let x be the first repeated node in $p_{1j}' \diamond p_{jn}^*$, and $p_{1j}' \diamond p_{jn}^*$ be the loopless $(1, n)$ -path obtained from $p_{1j}' \diamond p_{jn}^*$ after removal of that loop. Here p_{1x}' and p_{xn}^* correspond to p_{1j}' 's sub-path from 1 to x and p_{jn}^* 's sub path from x to n , respectively. Then, $p_{1x}' \diamond p_{xn}^*$ and $p_{1j}' \diamond p_{jn}^*$ have the same robustness cost. By hypothesis, all nodes in p_{jn}^* are associated with first labels, and so do the nodes in p_{xn}^* . If the same holds for p_{1x}' the result is proven. Otherwise the reasoning can be repeated. Because node labels are managed as a FIFO list, p_{1x}' has less nodes than p_{1j}' , and therefore repeating it a finite number of times leads to a contradiction, as expected. \square

From now on it will be assumed that labels are treated in a FIFO manner. Otherwise it should be verified whether a new one corresponds to a path with a loop.

The above dominance test is a pruning strategy adopted in [18], for the labels associated with any node. In the method introduced here, these tests are skipped for the labels z_n , and such a label is only selected when (6) holds with a strict inequality, according to Lemma 4.

A second pruning rule for the labels z_i , $i \in V \setminus \{n\}$, is inferred in Proposition 4. This property is based on a bounding condition satisfied by every sub-path of a robust shortest $(1, n)$ -path.

Proposition 4. *Let p be a robust shortest $(1, n)$ -path and p_{1i} be a $(1, i)$ -sub-path of p with label z_i , $i \in V$. Then,*

$$\max_{s \in S_k} \{z_i^s + LB_i^s\} \leq RC(p). \quad (7)$$

Proof. Let p_{1i} be a $(1, i)$ -path, $i \in V$, contained in a robust shortest $(1, n)$ -path p , and let z_i be its label. Then, $c^s(p) \geq c^s(p_{1i} \diamond p_{in}^{1,s}) = c^s(p_{1i}) + LB_i^s$, or, equivalently, $RD^s(p) \geq z_i^s + LB_i^s$, for every $s \in S_k$, and condition (7) is satisfied. \square

The second test for the labels z_i , $i \in V \setminus \{n\}$, allows to eliminate those that would produce $(1, n)$ -paths with robustness costs which are not better than the least computed value. In fact, denoting by UB an upper

bound for the optimum value of the problem, when

$$\max_{s \in S_k} \{z_i^s + LB_i^s\} \geq UB \quad (8)$$

holds with a strict inequality, then the $(1, i)$ -path associated with label z_i cannot be part of any optimal solution, and in case of equality, it can be part of an optimal solution with robustness cost UB . Nevertheless, taking into account that a candidate path with the same robustness cost UB is already known, z_i can be discarded in both cases. If (8) is satisfied with a strict inequality, this pruning rule is equivalent to the first one proposed in [18].

The value UB is initialized with the best robustness cost of the shortest paths for each scenario, keeping in mind that calculating their costs is fundamental to start the algorithm. Hence, UB is initialized with

$$\min_{s \in S_k} RC(p^{1,s}) = \min_{s \in S_k} \max_{r \in S_k} RD^r(p^{1,s}) = \min_{s \in S_k} \max_{r \in S_k} \{c^r(p^{1,s}) - LB_1^r\}. \quad (9)$$

This value is then updated as new labels z_n are computed.

The structure of the labeling algorithm for finding a robust shortest $(1, n)$ -path is described in the following.

Global algorithmic structure To start with, the computation of the trees \mathcal{T}^s of shortest (i, n) -paths and of the associate cost lower bounds LB_i^s are necessary, for each $s \in S_k$, $i \in V$. Any shortest path tree algorithm can be applied [1]. Then, the optimum cost upper bound UB is initialized with (9). In order to do that, calculating the deviation costs for the shortest paths over all scenarios is required. Since some of them can be the shortest for more than one scenario, the computation of their robustness costs can be avoided by using a list Q with only the distinct shortest paths. The first candidate is the path with the least robustness cost in Q .

A variable $RCaux$ is used to store the robustness cost of a $(1, n)$ -path associated with a label z_n being analyzed and it updates UB in case it improves the least robustness cost found so far. The variable sol represents the corresponding path, which is an optimal path candidate.

Another list X is used with the purpose of storing all the labels z_i to be scanned, $i \in V \setminus \{n\}$, under a FIFO policy. The scanned labels of node i which are not eliminated are stored in a list Z_i and the associate predecessor nodes are inserted in a list P_i . This list is used in order to retrieve the optimum solution at the end of the algorithm, by tracing back the nodes up to node 1.

A label z_i , $i \in V \setminus \{n\}$, is not discarded when it is not dominated by or equivalent to another label in Z_i and it satisfies $\max_{s \in S_k} \{z_i^s + LB_i^s\} < UB$. Then, all labels for node i in lists X and Z_i dominated by z_i must be removed, and, moreover, the corresponding predecessor nodes at P_i are deleted as well. Afterward, label z_i is inserted in lists X and Z_i for further evaluation, whereas i 's predecessor node is included in list P_i .

The pseudo-code for the labeling procedure is presented in Algorithm 1.

Computational time complexity order Algorithm 1 is pseudo-polynomial. The number of operations it performs is $\mathcal{O}(k^2n + k \max\{m, n^2W^2\})$ for any type of network, with W the maximum number of labels created for every node (details and calculations can be seen in Appendix).

Algorithm 1: Labeling approach for finding a robust shortest $(1, n)$ -path

```
1  $Q \leftarrow \emptyset$ ;  
2 for  $s = 1, \dots, k$  do  
3   Compute the tree  $\mathcal{T}^s$  in scenario  $s$ ;  
4    $Q \leftarrow Q \cup \{p^{1,s}\}$ ;  
5   for  $i = 1, \dots, n$  do  $LB_i^s \leftarrow c^s(p_{in}^{1,s})$  ;  
6  $UB \leftarrow \min\{RC(p^{1,s}) : s \in S_k\}$ ;  
7  $sol \leftarrow p^{1,s}$  such that  $RC(p^{1,s}) = UB$ ,  $s \in S_k$ ;  
8  $z_1 \leftarrow (-LB_1^1, \dots, -LB_1^k)$ ;  
9  $X \leftarrow \{z_1\}$ ;  $Z_1 \leftarrow \{z_1\}$ ;  
10 for  $i = 2, \dots, n - 1$  do  $Z_i \leftarrow \emptyset$ ;  $P_i \leftarrow \emptyset$  ;  
11 while  $X \neq \emptyset$  do  
12    $z_i \leftarrow$  first label in  $X$ ;  $X \leftarrow X - \{z_i\}$ ;  
13   for  $(i, j) \in A$  do  
14      $z_j \leftarrow (z_i^1 + c_{ij}^1, \dots, z_i^k + c_{ij}^k)$ ;  
15     if  $j = n$  then  
16        $RCaux \leftarrow \max\{z_j^s : s \in S_k\}$ ;  
17       if  $RCaux < UB$  then  
18          $UB \leftarrow RCaux$ ;  
19          $p_{1i} \leftarrow (1, i)$ -path traced back to node 1 by starting with  $P_i$ ;  
20          $sol \leftarrow p_{1i} \diamond \langle i, n \rangle$ ;  
21     else if  $z_j$  is not dominated by or equivalent to any label in  $Z_j$  and  $\max_{s \in S_k} \{z_j^s + LB_j^s\} < UB$   
22     then  
23       Delete from  $X$  and from  $Z_j$  all the labels of  $j$  that are dominated by  $z_j$ ;  
24       Delete from  $P_j$  the predecessor nodes associated with the labels deleted from  $Z_j$ ;  
25        $X \leftarrow X \cup \{z_j\}$ ;  $Z_j \leftarrow Z_j \cup \{z_j\}$ ;  $P_j \leftarrow P_j \cup \{i\}$ ;  
26 return  $sol$ 
```

3.2 Ranking algorithms

An alternative strategy for determining a loopless robust shortest path based on ranking loopless paths for a fixed scenario and combined with the definition of a cost upper bound is introduced in the following. This technique is inspired on the work of Dias and Clímaco [6], who considered the determination of a set of paths that are not dominated with respect to any possible scenario. With this goal they adapted the bicriteria shortest path algorithm by Clímaco and Martins [5]. This strategy was equally useful on continuous models for the same problem, after discretizing the costs interval data using simply their lower and upper bounds. For the robust shortest path problem with finite multi-scenarios, some adaptations to the previous method can be made, taking into account the computation of a loopless optimal solution and the new optimal values according to the number of scenarios involved. Namely, only loopless paths have to be ranked and the update of the costs upper bounds according to the least produced optimal values can be done till an optimal path is found. This defines a first version of the method. The second is enhanced by the application of reduction techniques similar to those presented for the labeling approach.

3.2.1 First version

As mentioned before, Lemma 3 and Proposition 1 provide the basis for computing a robust shortest path based on ranking loopless paths by non-decreasing order of their costs under a fixed scenario. Also, a

stopping criterion is imposed by means of a particular cost upper bound. Lemma 3 shows that there is a loopless solution, and therefore ranking loopless paths can be used to find it, even in networks with cycles. Additionally, Proposition 1 allows to establish an upper bound for the ranking, associated with the computed candidates to robust shortest path. Specifically, once a candidate loopless path p is returned by the ranking in scenario s , the search for other candidates consists of ranking loopless paths with a cost smaller than $c^s(p)$. In fact, the paths with exactly that cost will certainly have $RC(p)$ as their minimum robustness cost, and the goal of the algorithm is to find only one optimal solution. Since $s \in S(p)$, $RD^s(p) = RC(p)$ and, consequently, $c^s(p) = LB_1^s + RC(p)$. When this cost upper bound is set for any robust shortest path candidate p , it can be improved whenever a path q satisfying $RC(q) < RC(p)$ is found along the ranking. The next result shows this and that it is also possible to detect an exact solution when $s \in S(q)$.

Proposition 5. *Let $p \in P$ and $s \in S_k$. Let $q \in P \setminus \{p\}$ verify:*

1. $c^s(q) \leq LB_1^s + RC(p)$,
2. $RC(q) < RC(p)$,
3. $RC(\tilde{q}) \geq RC(p)$, $\forall \tilde{q} \in P \setminus \{q\} : c^s(\tilde{q}) < c^s(q)$.

Then, any robust shortest path \tilde{p} satisfies

$$c^s(q) \leq c^s(\tilde{p}) \leq LB_1^s + RC(q) < LB_1^s + RC(p). \quad (10)$$

Moreover, if condition $s \in S(q)$ also holds, then q is a robust shortest path as well.

Proof. Let $p \in P$, $s \in S_k$ and $q \in P \setminus \{p\}$, such that q satisfies conditions 1., 2. and 3.

By conditions 1. and 2., one has $LB_1^s + RC(q) < LB_1^s + RC(p)$. Besides, condition 2. and definition (2) allow to conclude that the robustness cost of a robust shortest path cannot exceed $RC(q)$. Therefore, the robust deviation in scenario s of every optimal solution \tilde{p} must satisfy $RD^s(\tilde{p}) \leq RC(q)$, which implies

$$\forall \tilde{p} \in P : \tilde{p} \text{ is a robust shortest path, } c^s(\tilde{p}) \leq RC(q) + LB_1^s < LB_1^s + RC(p). \quad (11)$$

Besides, by condition 3., every robust shortest path must have a cost greater than or equal to $c^s(q)$, in order to produce a robustness cost that does not exceed $RC(q)$, which implies

$$\forall \tilde{p} \in P : \tilde{p} \text{ is a robust shortest path, } c^s(\tilde{p}) \geq c^s(q). \quad (12)$$

From (11) and (12), one concludes (10).

In addition to 1., 2. and 3., let now be assumed that $s \in S(q)$. For this case, s is the scenario for which the robust deviation of q is maximum, which means that $RC(q) = RD^s(q)$. Thus, (10) becomes an equality, since $RC(q) + LB_1^s = RD^s(q) + LB_1^s = c^s(q)$, i.e.,

$$\forall \tilde{p} \in P : \tilde{p} \text{ is a robust shortest path, } c^s(\tilde{p}) = c^s(q). \quad (13)$$

On the other hand,

$$\forall \tilde{q} \in P \setminus \{q\} : c^s(\tilde{q}) = c^s(q), RC(\tilde{q}) \geq RD^s(\tilde{q}) = RD^s(q),$$

and, recalling that $RD^s(q) = RC(q)$, one infers that

$$\forall \tilde{q} \in P \setminus \{q\} : c^s(\tilde{q}) = c^s(q), RC(\tilde{q}) \geq RC(q).$$

Then, path q has the minimum robustness cost among the paths that satisfy (13), therefore it is a robust shortest path. \square

This result gives two necessary conditions concerning bounding the minimum robustness cost of a path when ranking in a scenario $s \in S_k$ and detecting a robust shortest path under specific assumptions. When considering the cost upper bound $LB_1^s + RC(p)$ for a loopless path p , if a loopless path q is found such that $RC(q) < RC(p)$, then, attending to (10), the upper bound can be improved to $LB_1^s + RC(q)$. Moreover, the analysis of the scenarios in which $RC(q)$ occurs, i.e. of the scenarios for which the robust deviation of q is maximum, and their identification with the ranking scenario is crucial to spare computational effort. In fact, the search can halt when $s \in S(q)$, because in this case it can be concluded that q is an optimal solution.

The efficiency of the method depends on the scenario in which the minimum robustness cost occurs versus the scenario for which the ranking is performed and on how many paths have to be computed after the shortest path in the ranking.

Analogously to Algorithm 1, the upper bound for the least robustness cost is initialized with the least robustness cost for the shortest paths over all scenarios, as in (9). Another important initialization issue concerns the choice of the scenario in which the ranking is performed. Without loss of generality, it will be chosen the scenario with the smallest index r for which the robustness cost of the first candidate optimal solution p^* , with $p^* = p^{1,u}$, for some $u \in S_k$, occurs, i.e.

$$r = \min \{s \in S_k : RD^s(p^*) = RC(p^*)\}. \quad (14)$$

Consequently, by Proposition 1, $c^r(p^*)$ is the first cost upper bound for the ranking in scenario r .

The structure of the algorithm for the robust shortest path problem based on ranking loopless paths is given in the following.

Global algorithmic structure The preliminary procedures of this approach are similar to Algorithm 1 and the variables Q , $RCaux$, UB and sol represent the same. Another variable stores the cost upper bound for the ranking, $Cmax$. The ranking scenario r is initially determined according to (14).

Several algorithms can be applied to rank loopless paths in general networks, for instance [13, 14, 19, 21]. For acyclic networks unconstrained ranking algorithms, generally more efficient, can be used, like [7, 14].

The list Q allows to control if some ranked path coincides with some shortest path $p^{1,s}$, $s \in S_k$, already analyzed, thus preventing its robustness cost from being recalculated. Whenever a ranked path $p^{l,r}$, $l \geq 1$, has a robustness cost $RCaux$ that is smaller than UB , the latter must be updated with $RCaux$ and the candidate optimal solution sol with $p^{l,r}$. Moreover, in case $I(p^{l,r}) = r$, the search halts, since $p^{l,r}$ is an optimal solution; otherwise the upper bound cost $Cmax$ is set to $LB_1^r + UB$.

The pseudo-code of the method just described is presented in Algorithm 2.

Computational time complexity order Algorithm 2 is of $\mathcal{O}(k^2n + k \max\{m, Ln\} + L \log L)$ time for acyclic networks and of $\mathcal{O}(k^2n + \max\{k, Ln\}(m + n \log n) + kLn)$ time for general networks, with L the number of ranked loopless paths (details and calculations can be seen in Appendix).

3.2.2 Hybrid version

The method presented in this section results from the combination between Algorithm 2 and some pruning techniques used in Algorithm 1. In order to apply these pruning rules in the broadest possible way, a specific ranking algorithm will be used, based on the deviation algorithm MPS, introduced in [14]. For completeness, first, the MPS method is very briefly reviewed. After that, the deviation algorithm used here is described

Algorithm 2: Ranking approach for finding a robust shortest $(1, n)$ -path

```

1  $Q \leftarrow \emptyset$ ;
2 for  $s = 1, \dots, k$  do
3    $p^{1,s} \leftarrow$  shortest path in scenario  $s$ ;
4    $Q \leftarrow Q \cup \{p^{1,s}\}$ ;
5  $UB \leftarrow \min\{RC(p^{1,s}) : s \in S_k\}$ ;
6  $sol \leftarrow p^{1,s}$  such that  $RC(p^{1,s}) = UB, s \in S_k$ ;
7  $r \leftarrow \min\{s \in S_k : RD^s(sol) = UB\}$ ;
8  $Cmax \leftarrow c^r(sol)$ ;
9  $l \leftarrow 2$ ;
10 while  $p^{l,r}$  exists do
11   Compute  $p^{l,r}$ ;
12   if  $c^r(p^{l,r}) \geq Cmax$  then break ;
13   if  $p^{l,r} \notin Q$  then
14      $RCaux \leftarrow RC(p^{l,r})$ ;
15     if  $RCaux < UB$  then
16        $UB \leftarrow RCaux$ ;
17        $sol \leftarrow p^{l,r}$ ;
18       if  $RD^r(p^{l,r}) = UB$  then break ;
19        $Cmax \leftarrow LB_1^r + UB$ ;
20    $j \leftarrow j + 1$ ;
21 return  $sol$ 

```

and the rules applied to discard useless paths are presented. Unless otherwise stated, it is assumed that the ranking is done with respect to a given scenario $r \in S_k$.

Let $p \in P$, $w \in V(p)$, and p_{1w} denote the $(1, w)$ -sub-path of p . The idea behind deviation algorithms for ranking paths, or loopless paths, is to generate l -th shortest path candidates, $l > 1$, as paths that coincide with p along p_{1w} and that deviate from p exactly at node w . Given that the aim of such methods is to rank paths by order of cost, one adds $(w, x) \in A$ and then the shortest (x, n) -path for scenario $r \in S_k$, according to Figure 3.(a). Hence, the generated paths have the form

$$q_{w,x}^{p,r} = p_{1w} \diamond \langle w, x \rangle \diamond p_{xn}^{1,r}, \quad x \in V^+(w), \quad (15)$$

where $V^+(w) = \{x \in V : (w, x) \in A\}$ is the set of head nodes of the arcs in G with tail node w . In this case p is called the father of $q_{w,x}^{p,r}$. Additionally, w and (w, x) are denominated the deviation node and the deviation arc of path $q_{w,x}^{p,r}$, respectively, and this path is said to be a deviation of p . When $w = 1$, p_{1w} reduces to the initial node, 1. By convenience, it is considered that the father of the shortest path in scenario r is not defined and that 1 is its deviation node.

Ranking paths in a certain scenario can be done either using the costs or the reduced costs. Thus, in order to decrease the number of performed operations in the MPS algorithm, the arc costs are replaced by reduced costs, as explained next. The reduced cost \bar{c}_{ij}^r of an arc $(i, j) \in A$ in scenario $r \in S_k$ is defined by

$$\bar{c}_{ij}^r = LB_j^r - LB_i^r + c_{ij}^r.$$

The reduced cost of a path $p \in P$ in scenario r is then given by

$$\bar{c}^r(p) = \sum_{(i,j) \in A(p)} \bar{c}_{ij}^r. \quad (16)$$

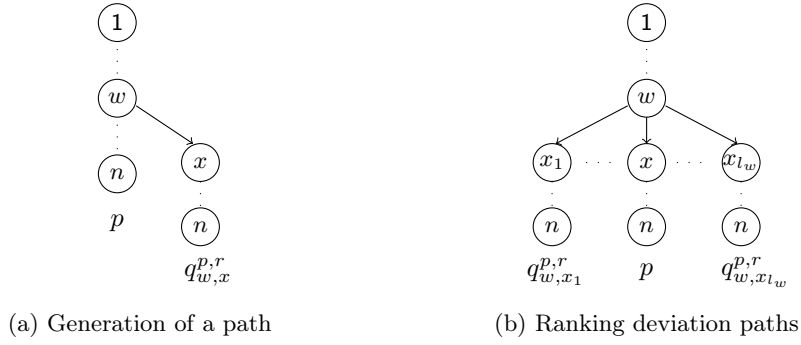


Figure 3: Deviation method

Now, because $\bar{c}_{ij}^r = 0$ for any $(i, j) \in \mathcal{T}^r$, then $\bar{c}^r(p_{in}^{1,r}) = 0$, for any $i \in V$. Hence, $\bar{c}^r(q_{w,x}^{p,r}) = \bar{c}^r(p_{1w}) + \bar{c}_{wx}^r$, $w \in V(p)$, $x \in V^+(w)$, and, therefore, the shortest path with form (15) with respect to scenario r contains the arc with the minimum reduced cost in $V^+(w)$.

Let $\widehat{V}^{+r}(w) = \{x_1, \dots, x_{l_w}\}$ represent the set $V^+(w)$ sorted by non-decreasing order of the reduced costs of the associate arcs with respect to scenario r , that is, such that $\bar{c}_{wx_1}^r \leq \dots \leq \bar{c}_{wx_{l_w}}^r$. Therefore, $\bar{c}^r(q_{w,x_1}^{p,r}) \leq \dots \leq \bar{c}^r(q_{w,x_{l_w}}^{p,r})$ and, thus, the costs in scenario r of the paths generated from a $(1, n)$ -path p by deviation at node $w \in V(p)$ are sorted, as Figure 3.(b) shows, in the following way:

$$c^r(q_{w,x_1}^{p,r}) \leq \dots \leq c^r(q_{w,x_{l_w}}^{p,r}). \quad (17)$$

Assuming that p_{1w} is a loopless $(1, w)$ -path, the deviation path $q_{w,x}^{p,r}$ results from the concatenation of three loopless paths and therefore it can still contain repeated nodes. However, the choice of x can be made in such a way that the least possible number of paths with loops is generated, by comparing the possible nodes x with the nodes in p_{1w} . Let $w \in V(p) \setminus \{n\}$ and $(w, x) \in A(p)$, the head nodes of the deviation arcs from path p at node w are chosen from the set:

$$V^{+r}(p_{1w}, x) = \{x_j \in \widehat{V}^{+r}(w) : x_j \neq x, \bar{c}_{wx_j}^r \geq \bar{c}_{wx}^r \text{ and } p_{1w} \diamond \langle w, x_j \rangle \text{ is loopless} \}. \quad (18)$$

The nodes considered for each path p are those from p 's deviation node to the node that precedes n . Figure 4.(a). shows a scheme of the deviation paths generated by the MPS algorithm with respect to a path with deviation arc (w, x) .

In the MPS algorithm the deviation from path p at a node w is obtained by taking the first head node in the ordered set $V^{+r}(p_{1w}, x)$. In order to simplify the choice of deviation arcs, the graph is stored in the sorted forward star form, that is, as mentioned earlier, each subset $\widehat{V}^{+r}(w)$ is sorted according to non-decreasing order of the reduced costs, for any $w \in V$ [14]. For scenario r , the MPS algorithm starts to generate deviations from the shortest path $p^{1,r}$ at every of its nodes but n . The resulting paths, one per each scanned node, are stored in a list and are selected, by non-decreasing order of the reduced costs, in future iterations. Each of these paths is identified as the l -th loopless shortest path with respect to scenario r in case it is loopless, for some $l > 1$. This process is repeated under the same conditions. Scanning only the nodes in loopless sub-paths reduces the calculation of paths containing loops, and selecting deviation arcs that have not been scanned earlier avoids the determination of repeated paths.

Any ranking strategy can be applied with Algorithm 2 in order to compute a robust shortest path. The hybrid algorithm here presented uses a specific variant of the MPS algorithm, as explained next. With this

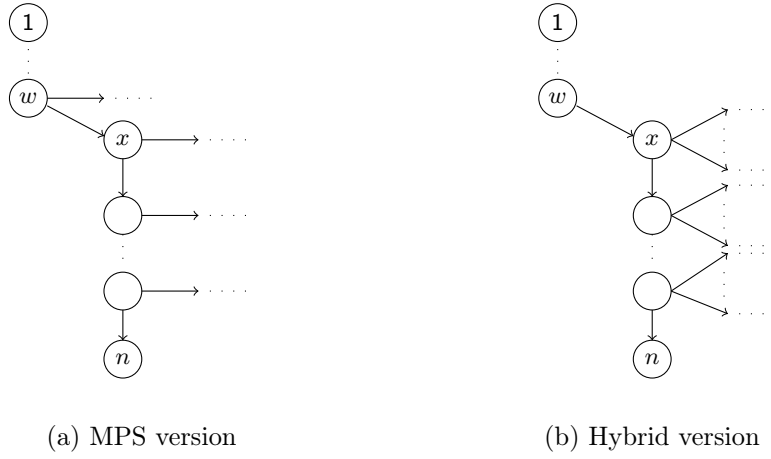


Figure 4: Deviation techniques used in the MPS and the hybrid algorithms

latter method, at most one new deviation path is generated when scanning a path node. The purpose is to avoid the calculation and the storage of unnecessary paths as much as possible, when ranking paths by order of cost. When looking for robust shortest paths using a ranking approach, the more solution candidates are generated, the higher the chances of computing paths with small robustness costs at an early stage. An expected consequence is to reduce faster the cost upper bound and to find an optimal solution quicker than when generating fewer candidates at a time. This technique is similar to the generalization of Yen's algorithm described in [14]. In the hybrid algorithm, the scanned nodes of a path p are those between its deviation node and the node that precedes n . Scanning one of those nodes, w , consists of generating all deviation paths of form (15), with the deviation arc (w, x_j) , with x_j chosen in the set $V^{+r}(p_{1w}, x)$, according to its underlying order. As shown in the following, this allows to apply and to extend the pruning rules used in Algorithm 1, and thus to discard some unnecessary deviation paths that do not lead to an optimal solution. Figure 4.(b) illustrates the deviation technique used in the hybrid algorithm with respect to a path with deviation arc (w, x) .

Corollary 1 presents the results in Propositions 1 and 4, rewritten using the notation introduced in the current section.

Corollary 1. *Let $p \in P$, $w \in V(p)$, $x \in V^+(w)$, and let $q_{w,x}^{p,r} = p_{1w} \diamond \langle w, x \rangle \diamond p_{xn}^{1,r}$ be the deviation path of p with deviation arc (w, x) . Let \tilde{p} be any robust shortest $(1, n)$ -path containing the sub-path p_{1w} . Then,*

1. $\max_{s \in S_k} \{c^s(p_{1w}) + LB_w^s - LB_1^s\} \leq RC(\tilde{p});$
2. $c^s(\tilde{p}) \leq c^s(q_{w,x}^{p,r}),$ for every $s \in S(q_{w,x}^{p,r}).$

The first point of this corollary is a sufficient condition for a deviation path to be a candidate to an optimal path. The second point states that the cost of a deviation path, in the scenario where its robustness cost occurs, is an upper bound for the cost of any robust shortest path containing the sub-path p_{1w} . These cost upper bounds can be combined with the ranking method previously described in order to obtain a deviation path which is a robust shortest path.

In the following, the bounds given by Corollary 1 are enhanced, taking into account the results used in Algorithm 2. Let $Cmax$ and UB denote upper bounds for the paths cost in scenario r and for the paths robustness cost, respectively. Like before, these values are initialized according to the least robustness cost

for the shortest paths in all scenarios $p^{1,s}$, $s \in S_k$. For a $(1, n)$ -path p and a node $w \in V(p)$, such that $(w, x_u) \in A(p)$ and $u < l_w$, every arc (w, x_j) , with $x_j \in V^{+r}(p_{1w}, x_u) = \{x_{u+1}, \dots, x_{l_w}\}$, is considered as a new deviation arc. Because the set $V^{+r}(p_{1w}, x_u)$ is sorted, the deviation paths $q_{w, x_{u+1}}^{p,r}, \dots, q_{w, x_{l_w}}^{p,r}$ satisfy condition (17). In this case, the following pruning rules apply:

1. By point 1. of Corollary 1, a sub-path p_{1w} does not produce robust shortest deviation paths if $\max_{s \in S_k} \{c^s(p_{1w}) + LB_w^s - LB_1^s\} > UB$. In this case, all the deviation paths $q_{w, x_j}^{p,r}$, with $u+1 \leq j \leq l_w$, can be skipped.
2. The rule above can be refined given that, by the same result, if $\max_{s \in S_k} \{c^s(p_{1w}) + c_{wx_j}^s + LB_{x_j}^s - LB_1^s\} > UB$, the path $p_{1w} \diamond \langle w, x_j \rangle \diamond p_{x_j n}^{1,r}$, $u+1 \leq j \leq l_w$, and its subsequent deviations will not lead to optimal paths, and thus can be skipped.
3. Let i be the first element in $\{u+1, \dots, l_w\}$ such that $c^r(q_{w, x_i}^{p,r}) > Cmax$. Then, all the deviation paths $q_{w, x_j}^{p,r}$, $i \leq j \leq l_w$, can be discarded.
4. Let $i' \in \{u+1, \dots, l_w\}$ be the smallest index such that $RC(q_{w, x_{i'}}^{p,r}) = RD^r(q_{w, x_{i'}}^{p,r}) \leq UB$. By point 2. of Corollary 1 all paths of form $q_{w, x_j}^{p,r}$, with $i' < j \leq l_w$, and subsequent deviations have a robustness cost not smaller than the optimum value. Therefore, they can be skipped.

The deviation process for the hybrid algorithm is performed for the first path, $p^{1,r}$, by scanning all its nodes but n and for the subsequent deviation paths of the form (15), by scanning all their nodes from the head node of their deviation arc till the one that precedes either n or the first which is repeated. Every deviation node is the tail node of an arc in \mathcal{T}^r . This is valid both for $p^{1,r}$, which is a path in that tree, as well as for the paths of the form (15), because they result from the concatenation with a path in that tree. Consequently, for any node $w \in V(p)$ that is scanned and $(w, x) \in A(p)$, it holds $\bar{c}_{wx}^r = 0$. Therefore, x is the first element in $\widehat{V}^{+r}(w)$, i.e. $x = x_1$, and the available head nodes of the deviation arcs with tail node w belong to $V^{+r}(p_{1w}, x_1)$. The obtained candidate paths are stored in a list X and the path with the least cost in scenario r is chosen to be analyzed in the next iteration. Throughout the algorithm, the upper bounds $Cmax$ and UB are updated and a loopless robust shortest path is identified when the stopping criterion used in Algorithm 2 is satisfied.

The steps of this method are described next.

Global algorithmic structure The preliminary procedures for this approach have points in common with both Algorithms 1 and 2. A list W stores the non discarded deviation paths in each iteration and another list X stores all such paths for all iterations. The variables $RCaux$, UB , sol and $Cmax$ have the same meaning as in Algorithm 2 and the ranking scenario r is determined in the same way. Additionally, the sorted forward star form of the network with respect to the costs in scenario r is obtained. The path being scanned is represented by variable p . The deviation nodes of new deviation paths are represented by w , the variable x_1 corresponds to the head node of p 's arc with tail node w , i.e. $p = p_{1w} \diamond \langle w, x_1 \rangle \diamond p_{x_1 n}^{1,r}$.

According to the pruning rules described above, one deviates at a given node w in case 1. or 2. are not satisfied and one may stop deviating at w when a deviation path $q_{w,x}^{p,r}$ satisfies 3. or 4. Whenever the robustness cost, $RCaux$, of a deviation path improves UB , the latter as well as the upper bound cost $Cmax$ are updated. Additionally, one can delete all the paths q in W which exceed the ranking bounds or that would not produce any optimal solution, i.e. that satisfy rules 2. or 3.

An iteration is complete once all the necessary nodes of path p have been scanned. Then, if this is a loopless path with robustness cost UB , it is identified as an optimal path candidate, and sol is updated. Additionally, the paths in X that satisfy the pruning rules 2. or 3. are removed from the list. Finally, all the paths stored in W are inserted in list X and the path to be considered at the next iteration is the shortest in X with respect to scenario r .

The pseudo-code of the second version of Algorithm 2 described above is presented in Algorithm 3.

Algorithm 3: Hybrid approach for finding a robust shortest $(1, n)$ -path

```

1  $Q \leftarrow \emptyset$ ;
2 for  $s = 1, \dots, k$  do
3   Compute the tree  $\mathcal{T}^s$  in scenario  $s$ ;
4    $Q \leftarrow Q \cup \{p^{1,s}\}$ ;
5   for  $i = 1, \dots, n$  do  $LB_i^s \leftarrow c^s(p_{in}^{1,s})$ ;
6  $UB \leftarrow \min\{RC(p^{1,s}) : s \in S_k\}$ ;
7  $sol \leftarrow p^{1,s}$  such that  $RC(p^{1,s}) = UB, s \in S_k$ ;
8  $r \leftarrow \min\{s \in S_k : RD^s(sol) = UB\}$ ;
9  $Cmax \leftarrow c^r(sol)$ ;
10  $X \leftarrow \emptyset; p \leftarrow p^{1,r}$ ;
11 Store the network in the sorted forward star form with respect to the costs for scenario  $r$ ;
12 while there exists a path  $p$  to be scanned such that  $RD^r(p) \neq UB$  do
13    $W \leftarrow \emptyset$ ;
14   for  $w \in V(p)$  from the head node of  $p$ 's deviation arc to the node that precedes  $n$  do
15      $p_{1w} \leftarrow (1, w)$ -sub-path of  $p$ ;
16     if  $p_{1w}$  is not loopless then break;
17     if  $\max_{s \in S_k} \{c^s(p_{1w}) + LB_w^s - LB_1^s\} > UB$  then break;
18      $x_1 \leftarrow$  head node of  $p$ 's arc with tail node  $w$ ;
19     for  $x \in \widehat{V}^{+r}(p_{1w}, x_1)$  do
20       if  $\max_{s \in S_k} \{c^s(p_{1w}) + c_{wx}^s + LB_x^s - LB_1^s\} \leq UB$  then
21          $q_{w,x}^{p,r} \leftarrow p_{1w} \diamond \langle w, x \rangle \diamond p_{xn}^{1,r}$ ;
22         if  $c^r(q_{w,x}^{p,r}) > Cmax$  then break;
23          $W \leftarrow W \cup \{q_{w,x}^{p,r}\}$ ;
24          $RCaux \leftarrow \max\{c^s(p_{1w}) + c_{wx}^s + LB_x^s - LB_1^s : s \in S_k\}$ ;
25         if  $RCaux < UB$  then
26            $UB \leftarrow RCaux; Cmax \leftarrow LB_1^r + UB$ ;
27           Delete from  $W$  any path  $q$  such that  $c^r(q) > Cmax$ , or
            $\max_{s \in S_k} \{c^s(q_{1w}) + c_{wx}^s + LB_x^s - LB_1^s\} > UB$ , with  $(w, x)$  the  $q$ 's deviation arc;
28         if  $RD^r(q_{w,x}^{p,r}) = UB$  then break;
29   if  $RC(p) = UB$  and  $p$  is loopless then  $sol \leftarrow p$ ;
30   Delete from  $X$  any path  $q$  such that  $c^r(q) > Cmax$ , or
    $\max_{s \in S_k} \{c^s(q_{1w}) + c_{wx}^s + LB_x^s - LB_1^s\} > UB$ , with  $(w, x)$  the  $q$ 's deviation arc;
31    $X \leftarrow X \cup W$ ;
32    $p \leftarrow$  shortest path for scenario  $r$  in  $X$ ;  $X \leftarrow X - \{p\}$ ;
33 return  $sol$ 

```

Computational time complexity order Like the previous methods, Algorithm 3 is pseudo-polynomial in terms of time. Its worst case complexity is of $\mathcal{O}(k^2n + kmH + m \log n)$ for acyclic networks and of $\mathcal{O}(k^2n + k \max\{mH, n \log n\} + m \log n)$ for general networks, with H the number of ranked deviation paths

(details and calculations can be seen in Appendix).

4 Example

Let $G(V, A, S_2)$ be the network depicted in Figure 5, and consider the application of Algorithms 1, 2 and 3 for finding a robust shortest $(1, 6)$ -path in $G(V, A, S_2)$.

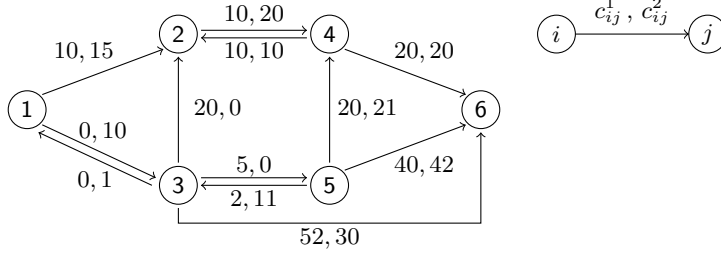


Figure 5: Network $G(V, A, S_2)$

The plots in Figure 6 show the tree of the shortest paths from every node to node 6 for scenario 1 – Figure 6.(a) – and the tree of the shortest paths from every node to node 6 for scenario 2 – Figure 6.(b). The values attached to a tree node i represent the cost of the $(i, 6)$ -path in that tree. For this example, one has $LB_1 = (40, 40)$, $LB_2 = (30, 40)$, $LB_3 = (40, 30)$, $LB_4 = (20, 20)$, $LB_5 = (40, 41)$.

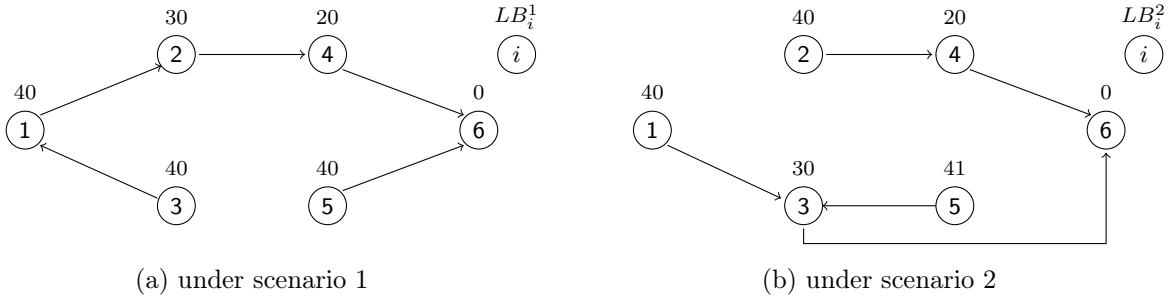


Figure 6: Shortest path trees rooted at $n = 6$ in $G(V, A, S_2)$

Initially, the elements of set Q are the shortest paths in scenarios 1 and 2, i.e. $p^{1,1} = \langle 1, 2, 4, 6 \rangle$, with $LB_1^1 = 40$, and $p^{1,2} = \langle 1, 3, 6 \rangle$, with $LB_1^2 = 40$, respectively. Because $c^1(p^{1,2}) = 52 < c^2(p^{1,1}) = 55$, $p^{1,2}$ is the path in Q with the minimum robustness cost, 12. This value initializes UB in Algorithms 1, 2 and 3. Thus, $p^{1,2}$ is initially set as a potential optimal solution, $sol = p^{1,2}$. Since $S(p^{1,2}) = \{1\}$, the loopless paths will be ranked in scenario 1 for Algorithms 2 and 3, and $Cmax = c^1(p^{1,2}) = 52$ is considered as the initial cost upper bound.

Application of Algorithm 1

Figure 7 shows the tree of paths that is obtained when applying Algorithm 1.

The method starts by selecting the label $z_1 = (-LB_1^1, -LB_1^2) = (-40, -40)$ and including it in Z_1 . When scanning z_1 , new labels for nodes 2 ($z_2 = (-30, -25)$) and 3 ($z_3 = (-40, -30)$) are generated. However, z_2 is discarded, given that $\max_{s \in S_2} \{z_2^s + LB_2^s\} = 15 > 12$ and thus it leads to paths with a robustness cost greater than 12. On the other hand, $\max_{s \in S_2} \{z_3^s + LB_3^s\} = 0 \leq 12$, which means that the extension of the

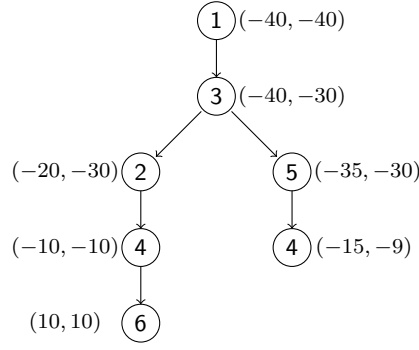


Figure 7: Search tree of paths produced by Algorithm 1

(1,3)-path associated with z_3 might be optimal. Then, z_3 is selected and inserted in Z_3 . After that, the labels $z_1 = (-40, -29)$, $z_2 = (-20, -30)$, $z_5 = (-35, -30)$ and $z_6 = (12, 0)$ are created. The first of them is dominated by label $(-40, -40)$ in Z_1 , therefore it is eliminated. Nevertheless, labels z_2 and z_5 are stored in Z_2 and Z_5 , respectively, because $\max_{s \in S_2} \{z_2^s + LB_2^s\} = 10 \leq 12$ and $\max_{s \in S_2} \{z_5^s + LB_5^s\} = 11 \leq 12$. The label z_6 satisfies $\max_{s \in S_2} \{z_6^s\} = 12$, hence it is discarded because it does not improve the best robustness cost obtained so far.

When selecting the label z_2 in Z_2 , $z_4 = (-10, -10)$ is created and, because $\max_{s \in S_2} \{z_4^s + LB_4^s\} = 10 \leq 12$, it is inserted in Z_4 . From label $z_5 \in Z_5$, the labels $z_3 = (-33, -19)$, $z_4 = (-15, -9)$ and $z_6 = (5, 12)$ are generated. The former label z_3 is dominated by $(-40, -30)$, so it is deleted; label z_6 is not stored either because it does not improve the robustness cost 12; the extensions of the path associated with z_4 can produce a robustness cost of at least $\max_{s \in S_2} \{z_4^s + LB_4^s\} = 11 \leq 12$, so z_4 is inserted in Z_4 . The two labels in Z_4 , $(-10, -10)$ and $(-15, -9)$, lead to $(10, 10)$ and $(5, 11)$ as labels of node 6, respectively. The first of them corresponds to the path $\langle 1, 3, 2, 4, 6 \rangle$ which has a least robustness cost of 10. This is the robust shortest path.

Application of Algorithm 2

Table 1 summarizes the steps of Algorithm 2 when applied to the network in Figure 5.

j	Path $p^{j,1}$	$c^1(p^{j,1})$	$c^2(p^{j,1})$	Updates
1	$\langle 1, 2, 4, 6 \rangle$	40	55	$UB \leftarrow 12$; $sol \leftarrow p^{1,2}$; $Cmax \leftarrow 52$
2	$\langle 1, 3, 5, 6 \rangle$	45	52	$RCaux \leftarrow 12$;
3	$\langle 1, 3, 5, 4, 6 \rangle$	45	51	$UB = RCaux \leftarrow 11 < 12$; $sol \leftarrow p^{3,1}$; $RD^1(p^{3,1}) \neq 11$; $Cmax \leftarrow 51$
4	$\langle 1, 3, 2, 4, 6 \rangle$	50	50	$UB = RCaux \leftarrow 10 < 11$; $sol \leftarrow p^{4,1}$; $RD^1(p^{4,1}) = 10$; Stop

Table 1: Simulation of Algorithm 2

The computation of the second shortest path in scenario 1, $p^{2,1} = \langle 1, 3, 5, 6 \rangle$, does not improve UB , which demands the calculation of path $p^{3,1} = \langle 1, 3, 5, 4, 6 \rangle$. This new path has a robustness cost smaller than the previous, 11, which allows to update UB . The new cost upper bound $Cmax = c^1(p^{3,1}) = 51$ for the ranking is also set and the potential optimal solution sol is updated with $p^{3,1}$. Since the maximum robust deviation of $p^{3,1}$ does not occur in scenario 1, the next path in the ranking must be obtained. Such path, $p^{4,1} = \langle 1, 3, 2, 4, 6 \rangle$, is the robust shortest (1,6)-path, because its robustness cost, 10, is the least UB obtained so far in scenario 1. The algorithm halts and $p^{4,1}$ is retrieved as the optimal solution.

Application of Algorithm 3

The tree of the paths obtained with Algorithm 3 is depicted in Figure 8.

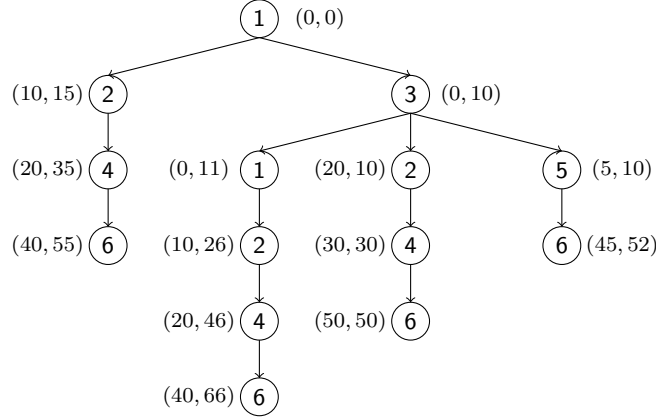


Figure 8: Tree of the deviation paths produced by Algorithm 3

The method starts by considering the same initial upper bounds as in Algorithm 2, $Cmax = 52$, $UB = 12$, and deviating from $p^{1,1} = \langle 1, 2, 4, 6 \rangle$ in all its nodes but 6, taking into account that

$$V^{+1}(\langle 1 \rangle, 2) = \{3\}; V^{+1}(\langle 1, 2 \rangle, 4) = \emptyset \text{ and } V^{+1}(\langle 1, 2, 4 \rangle, 6) = \emptyset.$$

The only path output on the first iteration of the method is $q_{1,2}^{p^{1,1},1} = \langle 1, 3, 1, 2, 4, 6 \rangle$, with $c^1(q_{1,2}^{p^{1,1},1}) = 40 < 52$, which contains a loop. This is the path p considered in the following iteration, and this time only node 3 is scanned. Besides, $\max_{s \in S_2} \{c^s(\langle 1, 3 \rangle) + LB_3^s - LB_1^s\} = 10 \leq 12$, thus potential optimal paths can deviate from p at node 3. Because $V^{+1}(\langle 1, 3 \rangle, 1) = \{5, 2, 6\}$, the arcs $(3, 5)$, $(3, 2)$ and $(3, 6)$ are considered as possible deviation arcs. When using deviation arc $(3, 5)$, the loopless path $q_{3,5}^{p,1} = \langle 1, 3, 5, 6 \rangle$ is computed, with $c^1(q_{3,5}^{p,1}) = 45 < 52$ and $RC(q_{3,5}^{p,1}) = 12$, which does not improve the best value obtained so far. Path $\langle 1, 3, 5, 6 \rangle$ is stored in list W and the deviation path $q_{3,2}^{p,1} = \langle 1, 3, 2, 4, 6 \rangle$ is obtained. This path satisfies $c^1(q_{3,2}^{p,1}) = 50 < 52$ and $RC(q_{3,2}^{p,1}) = 10 < 12$, so it is a new candidate for optimality and it is stored in W as well. Moreover, since $RD^1(q_{3,2}^{p,1}) = 10$, future deviations from path p at node 3 do not improve the best robustness cost obtained so far, and, therefore, the deviation arc $(3, 6)$ is not considered. In spite of $c^1(\langle 1, 3, 5, 6 \rangle) = 45 < 50$, path $\langle 1, 3, 5, 6 \rangle$ satisfies $\max_{s \in S_2} \{c^s(\langle 1, 3 \rangle) + c_{35}^s + LB_5^s - LB_1^s\} = 11 > 10$. Thus, this path will not produce optimal deviation paths and it is removed from W . Then, the only path left in W , $\langle 1, 3, 2, 4, 6 \rangle$, is transferred to the empty list X .

Because $V^{+1}(\langle 1, 3, 2 \rangle, 4) = V^{+1}(\langle 1, 3, 2, 4 \rangle, 6) = \emptyset$, no new deviation arcs exist and, therefore, $\langle 1, 3, 2, 4, 6 \rangle$ is the robust shortest path.

5 Computational experiments

This section is devoted to the empirical evaluation of the presented methods and their comparison with the exact algorithm by Yu and Yang [22]. This latter approach is based on dynamic programming, it applies a recursive relation to find the optimum value of the problem, starting with a cost upper bound for each scenario, given by the sum of the costs of the $n - 1$ arcs with the largest costs. This makes the method particularly sensitive to the range of cost values. From now on, Algorithms 1, 2 and 3 will be represented by LA , RA and HA , respectively, whereas YA will represent Yu and Yang's algorithm. In order to evaluate and compare the performances of these algorithms, they were implemented in Matlab 7.12. and ran on

a computer equipped with an Intel Pentium Dual CPU T2310 1.46GHz processor and 2GB of RAM. The implementations of LA , RA and HA use Dijkstra’s algorithm [1] to compute the shortest path tree \mathcal{T}^s for every scenario s . In LA , X is managed as a FIFO list. The MPS algorithm [14] is applied to rank the loopless paths in RA .

5.1 Input data and tests

The benchmarks used in the experiments correspond to randomly generated directed graphs with n nodes, m arcs and each arc cost assigned with a random real number in $U(0, b)$, $b \in \{20, 100\}$, for k scenarios. The computational tests were performed for $k \in \{2, 3, 4, 5, 10, 50, 100, 500, 1000, 5000\}$ scenarios and on

- complete networks, with $n \in \{5, 10, 15\}$,
- random networks, with $n \in \{5, 10\}$, $d \in \{2, 3\}$ and $n \in \{250, 500, 750\}$, $d \in \{5, 10, 15\}$, where $d = m/n$ represents the density.

For each network dimension of each type, 10 problems were generated and solved by the algorithms above.

5.2 Results

For each algorithm, the average and the standard deviation of the total CPU times (registered in seconds) are denoted by Ave_t and Std_t , respectively. The total CPU time is split in the time needed to compute the trees \mathcal{T}^s , $s \in S_k$, and the time required for the remaining procedures. The averages of the partial times are denoted by Ave_{t_i} , $i = 1, 2$. In terms of the total CPU time, $Ave_t = Ave_{t_1} + Ave_{t_2}$.

A first set of tests intended to compare the new methods with YA . These tests ran for networks with $n \in \{5, 10\}$, $d \in \{2, 3, 9\}$, $k \in \{2, 3\}$ and costs in $U(0, 20)$; the results are summarized on Table 2. For these cases, YA has a poor performance when compared with the other three methods, reporting results 10^4 or 10^5 times bigger than the corresponding ones for LA , RA and HA , which had all a similar behavior. Such difference increases with the number of scenarios. For $k \geq 4$ the tests for YA ran too slow, therefore the results are omitted.

n	d	k	LA		RA		HA		YA	
			Ave_t	Std_t	Ave_t	Std_t	Ave_t	Std_t	Ave_t	Std_t
5	2	2	0.008	0.002	0.008	0.002	0.007	0.002	14.101	2.658
		3	0.010	0.001	0.023	0.024	0.009	0.001	954.120	166.942
	3	2	0.010	0.004	0.009	0.002	0.009	0.005	19.490	1.526
		3	0.020	0.015	0.040	0.035	0.013	0.008	1673.298	230.015
	4	2	0.009	0.002	0.010	0.001	0.010	0.006	24.181	2.737
		3	0.017	0.022	0.026	0.026	0.017	0.015	2196.441	342.817
10	2	2	0.016	0.006	0.031	0.024	0.018	0.011	363.342	60.763
	3	2	0.014	0.001	0.017	0.007	0.013	0.002	513.922	57.470
	9	2	0.021	0.007	0.039	0.025	0.019	0.016	1305.751	76.355

Table 2: Averages and standard deviations of the total CPU times (in seconds)

The second set of tests considered bigger instances than the previous, costs generated in $U(0, 100)$, and the codes LA , RA and HA . The average partial running times are reported in Table 3 for complete networks and in Tables 6 and 7 for random networks. Analogously, Tables 4, 8 and 9 show the averages and the standard deviations of the total CPU times for complete and for random networks, respectively. Some of the values are omitted, when the codes were too slow.

n	k	LA		RA	HA
		$Ave_{t_1}^{(*)}$	Ave_{t_2}	Ave_{t_2}	Ave_{t_2}
5	2	0.007	0.003	0.005	0.002
	3	0.010	0.003	0.005	0.002
	4	0.018	0.003	0.007	0.002
	5	0.019	0.004	0.005	0.004
	10	0.028	0.005	0.006	0.004
	50	0.136	0.043	0.031	0.046
	100	0.229	0.148	0.918	0.158
	500	1.043	3.413	2.712	3.508
	1000	1.994	13.345	10.844	13.808
	5000	8.990	315.105	598.546	325.259
10	2	0.020	0.008	0.027	0.012
	3	0.022	0.006	0.072	0.004
	4	0.030	0.019	0.124	0.010
	5	0.038	0.008	0.718	0.006
	10	0.068	0.014	0.272	0.009
	50	0.264	0.056	0.696	0.064
	100	0.440	0.195	1.213	0.214
	500	2.174	4.240	5.456	4.384
	1000	4.167	16.965	17.959	17.441
	5000	21.399	421.006	909.843	427.992
15	2	0.019	0.030	0.855	0.011
	3	0.029	0.024	21.920	0.013
	4	0.041	0.021	83.914	0.011
	5	0.057	0.014	366.546	0.008
	10	0.084	0.014	175.924	0.015
	50	0.376	0.070	616.213	0.089
	100	0.663	0.234	1141.484	0.270
	500	2.981	5.003	1050.504	5.280
	1000	7.038	19.713	1136.937	20.486
	5000	31.923	524.525	3483.262	541.136

(*) : $Ave_{t_1}(LA) = Ave_{t_1}(RA) = Ave_{t_1}(HA)$

Table 3: Average partial CPU times for complete networks (in seconds)

Let Ave_r and Ave_h represent the average number of loopless $(1, n)$ -paths returned by the ranking in RA and HA , respectively, and Std_r and Std_h be the corresponding standard deviations. Such results are presented in Table 5, for complete networks, and in Table 10, for random networks. These measures are sufficient to determine the averages and the standard deviations of the total number of computed loopless paths, given by $Ave_{p_r} = Ave_r + k$ and $Ave_{p_h} = Ave_h + k$, respectively, and, analogously, the associate standard deviations by $Std_{p_r} = Std_r$ and $Std_{p_h} = Std_h$.

According to Tables 3, 6 and 7, computing the trees \mathcal{T}^s , $s \in S_k$, was the most demanding step in terms of time for codes LA and HA in most of the instances, except on some cases with many scenarios, like complete networks ($k \geq 1000$) or random networks with $n = 250$ and $k = 5000$. Nevertheless, LA presented other exceptions for random networks with few scenarios ($k \leq 5$).

In general, most of RA 's time was invested on the second step, namely when the number of ranked paths or the number of deviation costs demanded a major computational effort. The latter cases are reflected in the results obtained for all types of networks when $k \geq 100$ and the former stand for the denser networks, like complete networks with $n \in \{10, 15\}$ and random networks with $d = 15$. In fact, the higher the density of a network, the more arcs emerge from each node, which improves the chances of computing a large number of loopless paths till a solution is obtained, as Tables 5 and 10 show. Moreover, since Ave_r was always greater than Ave_h , $Ave_{t_2}(RA)$ was also always greater than $Ave_{t_2}(HA)$, even for the cases where the second phase had a minor role in the performance of RA . This was the case for complete networks with $n = 5$ and $k < 100$,

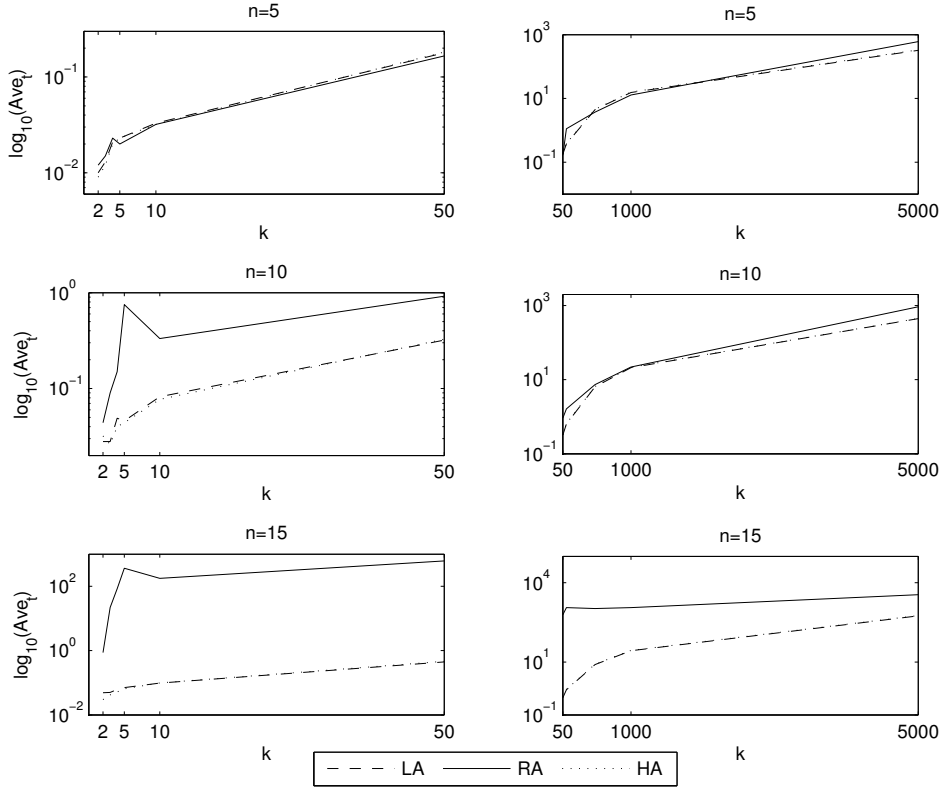


Figure 9: Average total CPU times for complete networks

where in average up to 3 loopless paths were ranked, and for random networks with small densities and few scenarios, as $d = 5$ and $k \in \{2, 3, 4\}$ or $d = 10$ and $k = 2$.

Code *HA* outperformed *RA* for all cases in terms of time, as shown in Tables 4, 8 and 9. Nevertheless, *HA* was not always the most efficient method, given that *LA* had the best performance in problems with $k \geq 100$. The standard deviations of the total CPU times provide information about the variability on the results towards the averages. In this sense, *LA* and *HA* were the most stable codes, with standard deviations generally smaller than the corresponding averages. Instead, *RA* had the most irregular performance due to the high values of $Std_t(RA)$, usually greater than $Ave_t(RA)$. This is supported by the high variability of the number of paths ranked by *RA* on Tables 5 and 10. In contrast, the number of paths ranked by *HA* did not vary much and the averages Ave_h are quite small, especially when $k \geq 50$ for denser networks, where only one iteration was needed to obtain the optimal solution.

The evolution of the average CPU times can be evaluated by varying a single parameter at a time. When n and d are fixed, Tables 3, 6 and 7 show that the average time for computing the trees \mathcal{T}^s , $s \in S_k$, grows when k increases, which is explained by the increase of the number of shortest paths ending at node n . In what concerns the second phase of the algorithms, $Ave_{t_2}(LA)$ and $Ave_{t_2}(HA)$ showed the smoothest growths. Instead, $Ave_{t_2}(RA)$ increased more irregularly with k , due to the unstable variation of Ave_r in Tables 5 and 10.

As computing the trees \mathcal{T}^s , $s \in S_k$, is the common initial task for all algorithms, their behavior on the second phase mimics the evolution of their average total CPU times. The plots in Figures 9 and 10 show these growths in logarithmic scale for the three codes, when k varies on complete networks with n fixed and

n	k	LA		RA		HA	
		Ave_t	Std_t	Ave_t	Std_t	Ave_t	Std_t
5	2	0.010	0.002	0.012	0.003	0.009	0.001
	3	0.013	0.001	0.015	0.004	0.012	0.002
	4	0.021	0.002	0.025	0.019	0.020	0.002
	5	0.023	0.002	0.024	0.009	0.023	0.003
	10	0.033	0.005	0.034	0.007	0.032	0.006
	50	0.179	0.008	0.167	0.028	0.182	0.011
	100	0.377	0.011	1.147	0.018	0.387	0.011
	500	4.456	0.038	3.755	0.700	4.551	0.024
	1000	15.339	0.105	12.838	0.735	15.802	0.122
5000	324.095	2.694	607.536	10.618	334.249	2.183	
10	2	0.028	0.008	0.047	0.047	0.032	0.030
	3	0.028	0.009	0.094	0.125	0.026	0.005
	4	0.049	0.016	0.154	0.189	0.040	0.018
	5	0.046	0.006	0.756	2.861	0.044	0.005
	10	0.082	0.022	0.340	0.420	0.077	0.009
	50	0.320	0.010	0.960	0.683	0.328	0.009
	100	0.635	0.006	1.653	1.508	0.654	0.022
	500	6.414	0.118	7.630	2.986	6.558	0.079
	1000	21.132	0.767	22.126	2.183	21.608	0.635
5000	442.405	11.154	931.242	52.261	449.391	8.616	
15	2	0.049	0.019	0.874	1.459	0.030	0.008
	3	0.053	0.018	21.949	75.275	0.042	0.017
	4	0.062	0.024	83.955	306.722	0.052	0.010
	5	0.071	0.021	366.603	895.563	0.065	0.007
	10	0.098	0.010	176.008	621.275	0.099	0.014
	50	0.446	0.024	616.589	1127.390	0.465	0.016
	100	0.897	0.019	1142.147	2323.143	0.933	0.017
	500	7.984	0.106	1053.485	2737.683	8.261	0.092
	1000	26.751	0.346	1143.975	2478.878	27.524	0.247
5000	556.448	21.708	3515.185	3472.625	573.059	27.474	

Table 4: Averages and standard deviations of the total CPU times for complete networks (in seconds)

on random networks with n and d fixed, respectively. The chosen density is 5 because these problems were solved till the end for all sizes, except when $k = 5000$ and $n = 750$. The averages $Ave_t(LA)$ and $Ave_t(HA)$ grew similarly with the increase of k but slower than $Ave_t(RA)$. The increase of the latter is steeper than for LA and HA and it is quite irregular for small values of k , due to the unsteady behavior of the ranking. Moreover, all averages increase slower when $k > 1000$, since all the performances become more dependent on the cost calculations.

k	$n = 5$				$n = 10$				$n = 15$			
	RA		HA		RA		HA		RA		HA	
	Ave_r	Std_r	Ave_h	Std_h	Ave_r	Std_r	Ave_h	Std_h	Ave_r	Std_r	Ave_h	Std_h
2	1	1	1	1	8	9	2	2	63	74	3	3
3	2	2	1	1	17	19	1	1	220	322	3	2
4	3	2	1	0	25	26	2	2	407	566	4	3
5	2	2	1	0	40	62	2	1	786	1209	2	1
10	3	2	1	0	40	33	1	0	757	1072	2	2
50	3	2	1	0	64	35	1	0	1436	1352	1	0
100	4	2	1	0	86	53	1	0	1923	1906	1	0
500	4	2	1	0	67	39	1	0	1400	1112	1	0
1000	4	2	1	0	81	47	1	0	1531	1151	1	0
5000	5	3	1	0	85	72	1	0	2086	1770	1	0

Table 5: Averages and standard deviations of the number of ranked loopless paths for complete networks

The obtained results may also be analyzed from the perspective of fixing the number of scenarios. Based on Figures 9 and 10, for different values of k , $Ave_t(LA)$ and $Ave_t(HA)$ become more distant from $Ave_t(RA)$ when n increases. This results from the growth of the number of paths in G with n , which may also affect

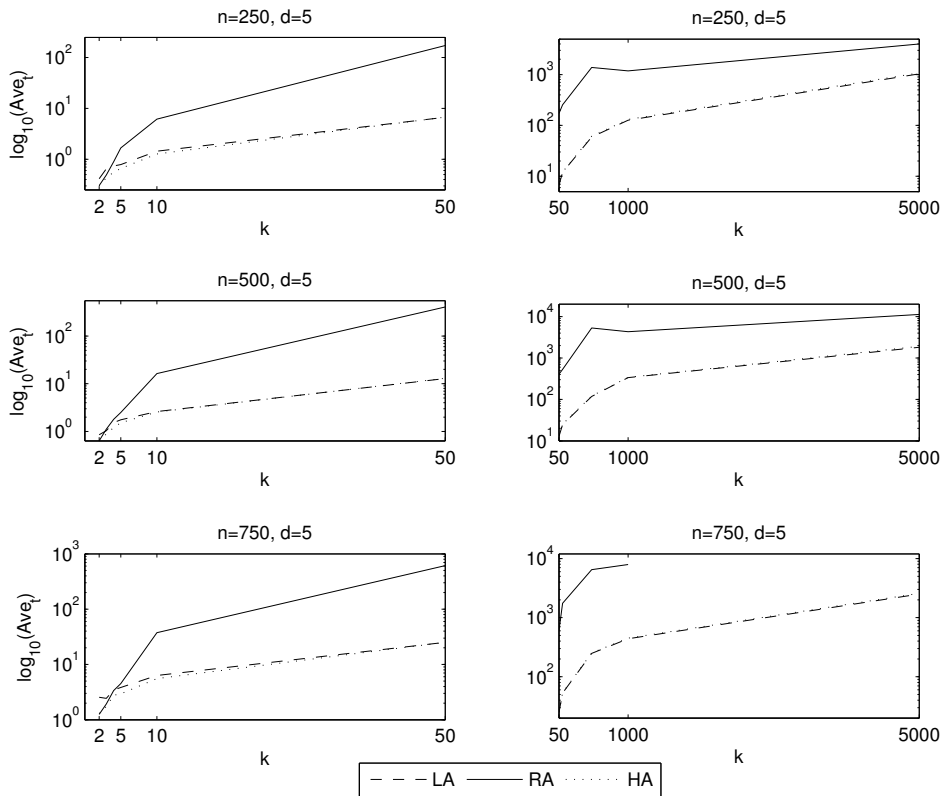


Figure 10: Average total CPU times for random networks with $d = 5$

their number of arcs and, consequently, the variety of paths, making the ranking heavier.

For random networks with fixed n and k , when d increases the number of paths and the average number of arcs emerging from each node increases too. This leads to a global growth on the total CPU times, specially for *RA*, as indicated by Tables 8 and 9, as well as by the plots in Figure 11 for random networks with $n = 250$. The graphics for random networks with $n \in \{500, 750\}$ are not included because the relative behavior of the codes in such cases is similar to those shown for $n = 250$.

In general, *LA* and *HA* were the most effective methods to solve the robust shortest path problem, showing similar behaviors. The experiments showed that *HA* was faster when the number of scenarios does not exceed 100, solving the problem in less than one minute in average, whereas *LA* was the best alternative for networks with 1000 or 5000 scenarios, running in less than one hour in average.

6 Conclusions

This work addressed the minmax regret robust shortest path problem with a finite number of scenarios. Some properties of this problem were derived and supported the development of three algorithms for finding an optimal loopless solution. The first algorithm is a labeling approach, the second is based on the ranking of loopless paths and the third is a hybrid version of the previous two that combines pruning techniques from both while ranking loopless paths in a specific manner. The novelty of the hybrid algorithm when compared to a simple version of the ranking based method is twofold. On the one hand, its pruning rules allow to skip uninteresting paths; while, on the other, it promotes the early generation of more candidate paths than with a standard implementation, seeking to produce good cost upper bounds. This strengthens the elimination

of bad solutions.

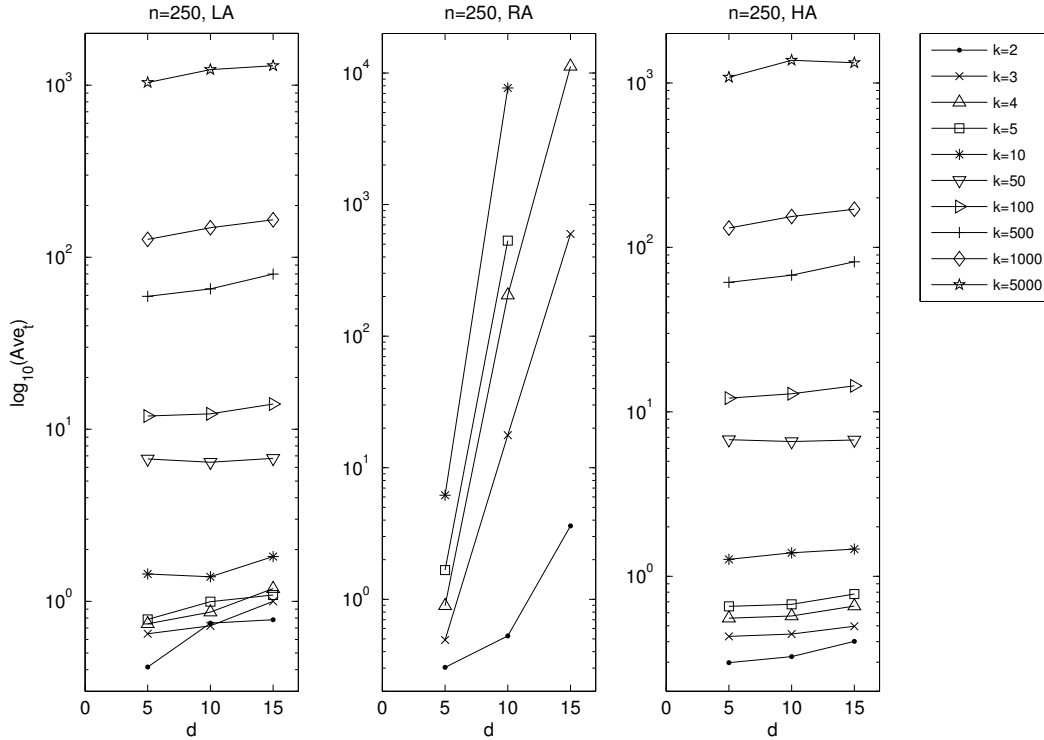


Figure 11: Average total CPU times for random networks with $n = 250$

The methods are pseudo-polynomial in terms of time and their worst case complexity orders depend on the model parameters, as well as on the number of labels for every node for the first method, and the number of ranked paths for the second and the third. Implementations of the three methods were tested on randomly generated networks and compared with the algorithm by Yu and Yang [22]. The new approaches were more efficient than Yu and Yang's algorithm. Besides, out of the three introduced methods, the ranking approach was the one with the poorest performance, due to the variable number of loopless paths that had to be ranked before the robust shortest path could be obtained. The changes introduced in the hybrid version resulted in an improvement of this step and, thus, of the initial version of the algorithm. The labeling and the hybrid methods had similar behaviors. Nevertheless, the former stood out only in problems with a large number of scenarios, while the latter was always the best when that number was small.

Future research on this subject can be directed to explore further techniques for reducing the network dimension of the model. Besides, because the time for solving the problem can be controlled for a relatively large number of scenarios, studying the relation with continuous cost models approximation theory is also suggested.

Acknowledgments This work has been partially supported by the Portuguese Foundation for Science and Technology under project grant PEst-C/EEI/UI308/2011. The research of Marisa Resende was partially funded by the Portuguese Foundation for Science and Technology under grant SFRH/BD/51169/2010.

Appendix

Computational times complexity

In order to determine the worst case computational complexities for the algorithms introduced earlier, some auxiliary procedures are analyzed in the following.

1. **Determination of a tree \mathcal{T}^s of shortest (i, n) -paths, $i \in V$, in a scenario $s \in S_k$, and of their costs for all scenarios:** The computational time complexity is $\mathcal{O}(m)$ for acyclic networks [1], and $\mathcal{O}(m + n \log n)$ for general networks, if using Fibonacci heaps [8]. Computing the costs for all scenarios has $\mathcal{O}(kn)$ in both cases, so this step has $\mathcal{O}(m + kn)$ for acyclic networks and $\mathcal{O}(m + n \log n + kn)$ for general networks.
2. **Calculation of the robustness cost of a shortest path $p^{1,s}$, $s \in S_k$, given LB_1^1, \dots, LB_1^k :** All the costs $c^r(p^{1,s})$, $r \in S_k$, were determined in 1. The k robust deviations, $RD^r(p^{1,s})$, $r \in S_k$, are obtained in $\mathcal{O}(k)$ time and their minimum can be found with $\mathcal{O}(k)$ comparisons. Consequently, the required work has time of $\mathcal{O}(k)$.
3. **Calculation of a label z_j from a given label z_i , with $(i, j) \in A$:** Label z_j is determined from z_i by adding the k costs of arc (i, j) , hence it is obtained in $\mathcal{O}(k)$ time.
4. **Dominance test between two labels:** Since in a worst case all the components of two labels are considered on a dominance test, at most k comparisons are involved and consequently this operation has $\mathcal{O}(k)$ time.

Algorithm 1 It is performed in two stages. The first one consists of the initialization steps, done in $\mathcal{O}(km + k^2n)$ for acyclic networks and in $\mathcal{O}(k(m + n \log n) + k^2n)$ for general networks, according to 1. In a worst case, the calculation of the robustness costs of all the k paths in Q is necessary, which takes $\mathcal{O}(k^2)$ time, attending to 2. Initializing the upper bound UB requires $\mathcal{O}(k)$ time. Hence, the total amount of operations that precede the generation of the labels is performed in $O_1^a = \mathcal{O}(km + k^2n)$ time for acyclic networks and $O_1^c = \mathcal{O}(k(m + n \log n) + k^2n)$ for general networks.

The second stage concerns the generation, scanning and pruning of the labels. Let W denote the maximum number of labels created for every node (a value dependent on the parameters n , m and k). Then, Wn is the number of iterations of the **while** loop in line 11 of Algorithm 1, and each of them implies at most n iterations of the **for** loop in line 13. In each of these iterations, the calculation of a new label is done in $\mathcal{O}(k)$ time, the dominance tests are performed in $\mathcal{O}(kW)$, and (8) is checked in $\mathcal{O}(1)$. Additionally, updating X , Z_i , P_i , $i \in V$, takes one operation, therefore, the second phase of Algorithm 1 has complexity of $\mathcal{O}(Wn^2kW) = \mathcal{O}(kn^2W^2)$.

Consequently, Algorithm 1 has a pseudo-polynomial time complexity of $\mathcal{O}(k^2n + k \max\{m, n^2W^2\})$ for any type of network, since $\log n \ll n$.

Algorithm 2 It has two major steps, the first concerned with preliminary procedures for the ranking and the second involving the ranking itself. Points 1. and 2. presented for the complexity analysis of Algorithm 1 are still valid regarding the complexity of the first phase of Algorithm 2. Thus, such procedures are performed in $O_1^a = \mathcal{O}(km + k^2n)$ time for acyclic networks and in $O_1^c = \mathcal{O}(k(m + n \log n) + k^2n)$ time for the general

case. According to (14), the choice of the scenario that defines the ranking is done in at most $\mathcal{O}(k)$ time, which does not affect the previous complexity bounds.

Regarding the second phase, if L loopless paths are ranked in scenario r , the time is of $\mathcal{O}(L \log L)$ for acyclic networks (after a initialization of $\mathcal{O}(m + n \log n)$), using Eppstein's algorithm [7], and of $\mathcal{O}(Ln(m + n \log n))$ in the general case, applying Yen's algorithm or one of its variants [13, 21]. Parameter L depends on n , m and k , and cannot be known in advance. In the worst case, the robustness costs of all the ranked paths, besides $p^{1,r}$, have to be determined. The cost of each of those paths in a given scenario can be computed in at most $\mathcal{O}(n)$ time, that is in $\mathcal{O}(kn)$ for all scenarios, resulting in $\mathcal{O}(Lkn)$ time complexity for all ranked paths. Therefore, the work required for the ranking and the related procedures can be done in $O_2^a = \mathcal{O}(L(\log L + kn))$ time for acyclic networks and in $O_2^c = \mathcal{O}(Ln(k + m + n \log n))$ time for general networks.

In conclusion, the time complexity of the algorithm is $O_1^a + O_2^a = \mathcal{O}(k^2n + k \max\{m, Ln\} + L \log L)$ for acyclic networks and $O_1^c + O_2^c = \mathcal{O}(k^2n + \max\{k, Ln\}(m + n \log n) + kLn)$ for general networks. Both bounds are pseudo-polynomial.

Algorithm 3 It has two phases. Like for the previous approaches the first phase, related with the preliminary procedures, can be performed in $\mathcal{O}_1^a = \mathcal{O}(km + k^2n)$ for acyclic networks and $\mathcal{O}_1^c = \mathcal{O}(k(m + n \log n) + k^2n)$ for general networks. Before the ranking starts, the arc costs are replaced by their reduced costs, $\mathcal{O}(m)$, and the network is represented in the sorted forward star form, $\mathcal{O}(m \log n)$ [14]. The second phase concerns the deviation process. Assume that H paths are ranked, that is, the **while** loop in line 12 is performed H times. In the worst case, scanning one path demands scanning all the network arcs, trying to generate new deviations and, for each, the costs of the father sub-path for all scenarios are obtained in $\mathcal{O}(kn)$ time. For a node w , the first test can be done in $\mathcal{O}(k)$ time. The second involves the analysis of the extension of each path $p_{1w} \diamond (w, x)$, with (w, x) the deviation arc, by checking the condition $\max_{s \in S_k} \{c^s(p_{1w}) + c_{wx}^s + LB_x^s - LB_1^s\} \leq UB$, which can be done in $\mathcal{O}(k)$ time. The third test involves the calculation of the cost of the deviation path $q_{w,x}^{p,r}$ under scenario r , in $\mathcal{O}(1)$. The fourth test implies the calculation of $RC(q_{w,x}^{p,r})$, of $\mathcal{O}(k)$, and its comparison with UB , of $\mathcal{O}(1)$. Hence, the total amount of work for each path is $\mathcal{O}(km)$, and the second phase has a complexity of $\mathcal{O}(Hkm)$. In these circumstances, the total complexity is $O_1^a + O_2^a = \mathcal{O}(k^2n + kmH + m \log n)$ for acyclic networks, and $O_1^c + O_2^c = \mathcal{O}(k^2n + k \max\{mH, n \log n\} + m \log n)$ for general networks. Like parameter L used in Algorithm 2, H depends on n , m and k and is unknown.

References

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows : Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [2] M.E. Bruni and F. Guerriero. An enhanced exact procedure for the absolute robust shortest path problem. *International Transactions in Operational Research*, 17:207–220, 2010.
- [3] A. Candia-Véjar, E. Álvarez Miranda, and N. Maculan. Minmax regret combinatorial optimization problems: An algorithmic perspective. *RAIRO Operations Research*, 45:101–129, 2011.

- [4] D. Catanzaro, M. Labbé, and M. Salazar-Neumann. Reduction approaches for robust shortest path problems. *Computers & Operations Research*, 38:1610–1619, 2011.
- [5] J. Clímaco and E. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.
- [6] L. Dias and J. Clímaco. Shortest path problems with partial information: Models and algorithms for detecting dominance. *European Journal of Operational Research*, 121:16–31, 2000.
- [7] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28:652–673, 1998.
- [8] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery*, 34:596–615, 1987.
- [9] V. Gabrel and C. Murat. Robust shortest path problems. Technical Report 7, Annales du LAMSADE, Paris, May 2007.
- [10] V. Gabrel, C. Murat, and A. Thiele. Recent advances in robust optimization and robustness: An overview, July 2012. Working paper. (www.optimization-online.org/DB_FILE/2012/07/3537.pdf).
- [11] O. E. Karasan, M. C. Pinar, and H. Yaman. The robust shortest path problem with interval data. Technical report, Bilkent University, Ankara, Turkey, 2001.
- [12] P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Kluwer Academic Publishers, Boston, 1997.
- [13] E. Martins and M. Pascoal. A new implementation of yen’s ranking loopless paths algorithm. *4OR – Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1:121–134, 2003.
- [14] E. Martins, M. Pascoal, and J. Santos. Deviation algorithms for ranking shortest paths. *The International Journal of Foundations of Computer Science*, 10:247–263, 1999.
- [15] R. Montemanni and L. Gambardella. An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31:1667–1680, 2004.
- [16] R. Montemanni and L. Gambardella. The robust shortest path problem with interval data via Benders decomposition. *4OR – Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 3:315–328, 2005.
- [17] R. Montemanni, L. Gambardella, and V. Donati. A branch and bound algorithm for the robust shortest path problem with interval data. *Operations Research Letters*, 32:225–232, 2004.
- [18] I. Murthy and S.-S. Her. Solving min-max shortest-path problems on a network. *Naval Research Logistics*, 39:669–683, 1992.
- [19] M. Pascoal. Implementations and empirical comparison for K shortest loopless path algorithms. In *Online Proc. of The Ninth DIMACS Implementation Challenge: The Shortest Path Problem*. DIMACS, USA, November 2006.

- [20] P. Perny and O. Spanjaard. An axiomatic approach to robustness in search problems with multiple scenarios. In *Proceedings of the 19th conference on Uncertainty in Artificial Intelligence*, pages 469–476. Acapulco, Mexico, 2003.
- [21] J. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.
- [22] G. Yu and J. Yang. On the robust shortest path problem. *Computers Operations Research*, 25:457–468, 1998.

n	d	k	LA		RA	HA
			$Ave_{t_1}^{(*)}$	Ave_{t_2}	Ave_{t_2}	Ave_{t_2}
250	5	2	0.243	0.172	0.061	0.055
		3	0.374	0.274	0.117	0.057
		4	0.487	0.253	0.407	0.069
		5	0.580	0.205	1.085	0.077
		10	1.152	0.290	5.024	0.117
		50	6.239	0.473	166.560	0.528
		100	11.052	0.899	243.745	1.074
		500	51.618	7.662	1332.511	9.734
		1000	102.774	24.211	1078.863	28.347
	5000	514.241	522.163	3480.725	568.822	
	10	2	0.268	0.479	0.259	0.057
		3	0.371	0.350	17.300	0.074
		4	0.489	0.376	204.664	0.084
		5	0.587	0.408	531.604	0.089
		10	1.242	0.147	7694.775	0.146
		50	5.883	0.554	–	0.712
		100	11.238	1.060	–	1.633
		500	56.614	8.883	–	11.199
		1000	120.067	28.493	–	33.852
	5000	577.927	655.516	–	795.650	
	15	2	0.328	0.453	3.284	0.074
		3	0.407	0.591	598.616	0.090
		4	0.547	0.639	11256.692	0.112
		5	0.668	0.420	–	0.111
		10	1.273	0.548	–	0.190
		50	6.066	0.710	–	0.673
		100	13.021	1.000	–	1.381
500		69.681	10.187	–	12.077	
1000		135.152	29.934	–	35.643	
5000	599.005	698.254	–	727.915		
500	5	2	0.486	0.368	0.152	0.148
		3	0.718	0.322	0.401	0.166
		4	1.022	0.506	0.767	0.180
		5	1.316	0.433	1.179	0.208
		10	2.276	0.340	14.091	0.268
		50	11.990	0.876	393.408	1.013
		100	23.050	1.607	508.091	2.228
		500	108.264	10.247	5177.183	13.169
		1000	301.306	33.695	3999.946	39.847
	5000	1190.716	631.577	10060.447	719.241	
	10	2	0.599	0.764	0.428	0.170
		3	0.855	0.720	31.129	0.198
		4	1.084	0.673	491.860	0.236
		5	1.449	1.063	3625.298	0.282
		10	2.898	0.639	–	0.381
		50	13.586	1.288	–	1.523
		100	28.917	2.718	–	3.778
		500	130.719	13.519	–	18.441
		1000	316.801	38.556	–	49.188
	5000	1346.546	707.249	–	743.674	
	15	2	0.586	1.461	2.769	0.175
		3	0.909	2.028	2209.236	0.245
		4	1.151	1.842	–	0.343
		5	1.447	1.223	–	0.337
		10	2.839	1.572	–	0.581
		50	15.486	1.073	–	1.458
		100	36.128	2.308	–	2.981
500		165.870	14.892	–	20.634	
1000		332.505	42.902	–	53.244	
5000	1325.300	746.045	–	783.147		

(*) : $Ave_{t_1}(LA) = Ave_{t_1}(RA) = Ave_{t_1}(HA)$

Table 6: Average partial CPU times for random networks with $n \in \{250, 500\}$ (in seconds)

n	d	k	LA		RA	HA
			$Ave_{t_1}^{(*)}$	Ave_{t_2}	Ave_{t_2}	Ave_{t_2}
750	5	2	0.967	1.582	0.380	0.329
		3	1.393	1.051	0.537	0.348
		4	2.352	1.117	1.014	0.430
		5	2.492	1.346	2.015	0.443
		10	4.914	1.402	32.518	0.634
		50	23.491	1.562	588.175	1.629
		100	50.008	2.768	1695.842	3.727
		500	233.830	14.968	6256.625	19.879
		1000	397.955	40.415	7532.278	51.006
	5000	1873.489	627.860	–	712.522	
	10	2	0.822	1.740	0.597	0.321
		3	1.491	2.357	46.015	0.390
		4	1.978	2.496	312.169	0.433
		5	2.434	3.189	3087.081	0.526
		10	4.239	2.195	–	0.659
		50	21.585	3.553	–	3.617
		100	41.405	4.681	–	6.179
		500	215.853	17.708	–	26.255
		1000	462.043	52.427	–	73.979
	5000	2041.511	824.071	–	977.557	
	15	2	0.900	2.665	12.107	0.351
		3	1.422	3.477	2195.500	0.385
		4	1.740	3.385	–	0.461
		5	2.140	3.990	–	0.502
		10	4.447	3.356	–	0.752
		50	26.819	1.838	–	2.150
		100	45.036	2.670	–	4.088
		500	264.081	16.705	–	23.343
		1000	497.384	44.877	–	65.762
	5000	2226.412	1093.321	–	1343.007	

(*) : $Ave_{t_1}(LA) = Ave_{t_1}(RA) = Ave_{t_1}(HA)$

Table 7: Average partial CPU times for random networks with $n = 750$ (in seconds)

n	d	k	LA		RA		HA		
			Ave_t	Std_t	Ave_t	Std_t	Ave_t	Std_t	
250	5	2	0.415	0.051	0.304	0.041	0.298	0.031	
		3	0.648	0.208	0.491	0.061	0.431	0.106	
		4	0.740	0.083	0.894	0.669	0.556	0.023	
		5	0.785	0.078	1.665	1.345	0.657	0.018	
		10	1.442	0.179	6.176	6.459	1.269	0.082	
		50	6.712	0.558	172.799	305.654	6.767	0.974	
		100	11.951	0.572	254.797	325.766	12.126	0.703	
		500	59.280	2.572	1384.129	2837.640	61.352	2.140	
		5000	126.985	4.327	1181.637	1818.132	131.121	6.917	
			5000	1036.404	54.742	3994.966	3619.199	1083.063	59.052
		10	2	0.747	0.335	0.527	0.606	0.325	0.036
	3		0.721	0.212	17.671	60.649	0.445	0.030	
	4		0.865	0.292	205.153	407.848	0.573	0.016	
	5		0.995	0.317	532.191	1902.340	0.676	0.023	
	10		1.389	0.061	7696.017	15783.047	1.388	0.023	
	50		6.437	0.380	–	–	6.595	0.271	
	100		12.298	0.579	–	–	12.871	1.190	
	500		65.497	3.159	–	–	67.813	3.877	
	5000		148.560	11.672	–	–	153.919	9.636	
			5000	1233.443	114.045	–	–	1373.577	318.609
		15	2	0.781	0.416	3.612	5.398	0.402	0.020
	3		0.998	0.375	599.023	1526.844	0.497	0.036	
	4		1.186	0.491	11257.239	18661.837	0.659	0.061	
	5		1.088	0.387	–	–	0.779	0.037	
	10		1.821	0.553	–	–	1.463	0.046	
	50		6.776	0.828	–	–	6.739	0.310	
	100		14.021	1.224	–	–	14.402	2.065	
	500		79.868	7.349	–	–	81.758	5.028	
5000	165.086		11.056	–	–	170.795	10.231		
		5000	1297.259	71.989	–	–	1326.920	78.640	
500	5	2	0.854	0.209	0.638	0.073	0.634	0.010	
		3	1.040	0.085	1.119	0.616	0.884	0.177	
		4	1.528	0.439	1.789	0.592	1.202	0.093	
		5	1.749	0.318	2.495	0.923	1.524	0.195	
		10	2.616	0.144	16.367	26.046	2.544	0.104	
		50	12.866	1.275	405.398	494.703	13.003	0.470	
		100	24.657	1.313	531.141	689.872	25.278	2.107	
		500	118.511	3.921	5285.447	7634.751	121.433	6.216	
		5000	335.001	48.053	4301.252	5678.683	341.153	30.030	
			5000	1822.293	77.472	11251.163	13902.215	1909.957	141.642
		10	2	1.363	0.566	1.027	0.729	0.769	0.061
	3		1.575	0.675	31.984	124.389	1.053	0.045	
	4		1.757	0.696	492.944	996.288	1.320	0.065	
	5		2.512	1.149	3626.747	14133.284	1.731	0.099	
	10		3.537	0.750	–	–	3.279	0.167	
	50		14.874	1.741	–	–	15.109	1.682	
	100		31.635	4.991	–	–	32.695	4.511	
	500		144.238	19.587	–	–	149.160	15.950	
	5000		355.357	19.211	–	–	365.989	28.429	
			5000	2053.795	208.749	–	–	2090.220	166.270
		15	2	2.047	0.916	3.355	7.932	0.761	0.064
	3		2.937	1.057	2210.145	8355.208	1.154	0.210	
	4		2.993	1.492	–	–	1.494	0.179	
	5		2.670	0.713	–	–	1.784	0.075	
	10		4.411	2.060	–	–	3.420	0.745	
	50		16.559	1.803	–	–	16.944	1.632	
	100		38.436	6.725	–	–	39.109	4.862	
	500		180.762	17.927	–	–	186.504	13.274	
5000	375.407		32.030	–	–	385.749	33.258		
		5000	2071.345	105.449	–	–	2108.447	88.095	

Table 8: Averages and standard deviations of the total CPU times for random networks with $n \in \{250, 500\}$ (in seconds)

n	d	k	LA		RA		HA	
			Ave_t	Std_t	Ave_t	Std_t	Ave_t	Std_t
750	5	2	2.549	0.760	1.347	0.099	1.296	0.041
		3	2.444	0.343	1.930	0.231	1.741	0.034
		4	3.469	1.897	3.366	0.774	2.782	0.239
		5	3.838	0.829	4.507	1.615	2.935	0.212
		10	6.316	0.934	37.432	69.502	5.548	0.709
		50	25.053	2.515	611.666	655.160	25.120	2.481
		100	52.776	6.762	1745.850	3676.003	53.735	4.564
		500	248.798	15.556	6490.455	10420.115	253.709	20.977
		1000	438.370	42.543	7930.233	10162.621	448.961	31.549
	5000	2501.349	144.605	—	—	2586.011	220.335	
	10	2	2.562	1.261	1.419	0.546	1.143	0.061
		3	3.848	1.298	47.506	174.138	1.881	0.356
		4	4.474	1.578	314.147	389.063	2.411	0.240
		5	5.623	2.014	3089.515	4193.368	2.960	0.410
		10	6.434	1.929	—	—	4.898	0.310
		50	25.138	3.724	—	—	25.202	3.183
		100	46.086	4.178	—	—	47.584	5.489
		500	233.561	26.755	—	—	242.108	21.534
		1000	514.470	40.275	—	—	536.022	56.891
	5000	2865.582	240.516	—	—	3019.068	430.135	
	15	2	3.565	2.051	15.672	41.596	1.251	0.062
		3	4.899	2.633	2196.922	5206.206	1.807	0.070
		4	5.125	2.690	—	—	2.201	0.198
		5	6.130	2.631	—	—	2.642	0.266
		10	7.803	3.498	—	—	5.199	0.325
		50	28.657	3.286	—	—	28.969	4.180
		100	47.706	3.610	—	—	49.124	3.000
		500	280.786	7.637	—	—	287.424	21.962
		1000	542.261	42.913	—	—	563.146	72.547
	5000	3319.733	623.843	—	—	3569.419	797.726	

Table 9: Averages and standard deviations of the total CPU times for random networks with $n = 750$ (in seconds)

		$n = 250$				$n = 500$				$n = 750$			
d	k	RA		HA		RA		HA		RA		HA	
		Ave_r	Std_r	Ave_h	Std_h	Ave_r	Std_r	Ave_h	Std_h	Ave_r	Std_r	Ave_h	Std_h
5	2	18	15	2	2	23	17	1	0	26	22	2	1
	3	42	23	2	1	75	63	3	3	108	58	4	3
	4	88	54	3	3	144	82	2	2	178	105	3	1
	5	162	81	1	1	192	116	3	3	265	151	3	2
	10	330	193	3	2	565	379	4	2	724	534	4	4
	50	1193	1105	6	10	2296	1805	16	28	3050	2329	11	27
	100	1449	1526	8	14	2581	2363	25	32	3843	4212	31	38
	500	2517	2786	22	38	6483	6334	36	62	7580	8616	34	47
	1000	2710	2914	34	49	5629	5706	11	29	7745	7919	107	123
5000	2379	2799	54	59	6822	7334	63	90	—	—	34	53	
10	2	41	39	4	2	46	56	4	4	48	40	7	10
	3	282	246	7	10	307	336	10	7	375	439	10	8
	4	859	890	6	4	1354	989	10	11	1532	728	13	19
	5	1227	1376	5	5	2431	2842	15	13	4149	2587	18	19
	10	5263	5230	6	6	—	—	9	10	—	—	35	43
	50	—	—	14	40	—	—	35	108	—	—	134	221
	100	—	—	23	70	—	—	74	154	—	—	154	249
	500	—	—	24	74	—	—	44	136	—	—	39	120
	1000	—	—	23	71	—	—	28	86	—	—	120	253
5000	—	—	41	84	—	—	1	0	—	—	106	236	
15	2	143	139	4	5	106	144	6	3	180	224	10	12
	3	1254	1125	9	7	1725	2288	15	12	2085	2669	16	9
	4	5478	5008	13	10	—	—	36	26	—	—	23	23
	5	—	—	11	11	—	—	26	19	—	—	28	17
	10	—	—	12	13	—	—	47	76	—	—	41	22
	50	—	—	1	0	—	—	1	0	—	—	1	0
	100	—	—	1	0	—	—	1	0	—	—	1	0
	500	—	—	1	0	—	—	1	0	—	—	1	0
	1000	—	—	1	0	—	—	1	0	—	—	1	0
5000	—	—	1	0	—	—	1	0	—	—	1	0	

Table 10: Averages and standard deviations of the number of ranked loopless paths for random networks