Instituto de Engenharia de Sistemas e Computadores de Coimbra
Institute of Systems Engineering and Computers
INESC – Coimbra

Marta Pascoal and Marisa Resende

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

No. 6 2014

ISSN: 1645-2631

Instituto de Engenharia de Sistemas e Computadores de Coimbra
INESC – Coimbra
Rua Antero de Quental, 199; 3000 - 033 Coimbra; Portugal
www.inescc.pt

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

Marta M. B. Pascoal*, Marisa Resende

September, 2013

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. These models provide partial information associated with a network and assume a finite number of parameters for the arc costs. These models have been analyzed under a multi-criteria context for the shortest path problem and used as a tool to solve robust shortest path problems on interval data models when the solution set of scenarios can be discretized. This work addresses the robust shortest path problem with a minmax regret objective function on a finite multi-scenario model. This leads to the determination of an optimum path in the sense that it has the minimum maximum deviation from the shortest one over all scenarios. Some properties of the problem and of its optimum solutions are derived. These results allow to introduce three approaches, one labeling algorithm, an algorithm based on ranking loopless paths, and another that ranks loopless paths in a suitable way to apply the early elimination of useless solutions. The algorithms are tested on random networks, and the obtained computational results are reported and discussed.

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

1 Introduction

The determination of a shortest path connecting two nodes of a network is a well known problem that consists in finding a path with the least cost, assuming that each arc is associated with a single and deterministic value. When trying to model reality these cost values are not always known or sometimes they carry some inaccuracy. The work by Dias and ClÃmaco [6] assumes that only partial information is known about the problem, addresses the shortest path problem from a multicriteria and a decision making points of view and presents algorithms to compute the set of non dominated paths taking into account the available information, under certain conditions. Perny and Spanjaard [20] adopt the same type of assumption and deal with more general network optimization problems. Another approach that has been used to cope with uncertainty is robust optimization [12]. In these cases several scenarios are considered for the arcs cost, and two uncertainty models have been considered, interval data models and discrete data set models. On the first of these models

*Corresponding author

the costs vary within given intervals, whereas on the second the costs belong to discrete finite sets. When the information given in different scenarios is aggregated, several criteria can be used. One of the most common aims at evaluating and 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 the minmax shortest path problem or the absolute robust shortest path problem, consists of finding the path with the minimum maximum cost over all scenarios. The second, known as the minmax regret robust shortest path problem or the 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.

The robust shortest path problem with a discrete set of scenarios was initially proposed by Yu and Yang [22]. This work introduced pseudo-polynomial algorithms for the problem, based on dynamic programming, as well as a more specific method designed for layered networks. It was shown that the problem is strongly NP-hard if the number of scenarios is unlimited, and then a heuristic is developed to compute an approximate optimal solution. Murthy and Her [18] studied the absolute version of the problem with a finite set of scenarios and introduced a labeling algorithm based on a multicriteria approach. The method includes a dominance test between labels as well as pruning techniques developed in order to discard some of them. Later, Bruni and Guerriero [2] tested heuristic rules for guiding the search performed by Murthy and Her's algorithm.

Other works have focused a similar problem but consider 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 focused the case of acyclic networks [11]. In this paper a finite number of scenarios containing the optimum solution could be determined by conjugating the upper and lower limits of the cost intervals. Several new exact techniques were developed later by Montemanni et al. which extended the determination of a robust shortest paths in general networks [15, 16, 17]. More recently, Catanzaro, LabbÃ© and Salazar-Neumann proposed reduction tools for preprocessing a network, in order to speed up the computational effort on solving the interval data robust shortest path problem [4]. Recent surveys on these topics can be found in works like [3, 9, 10].

The present work adopts the minmax regret as the robustness criterion according to a finite multi-scenario model, and considers the determination of an optimum path between a given pair of nodes in a network. Three exact methods are developed for computing a robust shortest path problem. The first is a labeling algorithm including cost lower and upper bounds to discard non interesting 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 solutions determined along the process. Finally, the third is also a ranking approach enhanced with cost bounds analogous to the first method, and designed with the purpose of fostering the generation of new paths and of associate cost bounds that 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 the associate cost. Section 3 is devoted to the algorithms presentation and to their worst case complexity analysis. In Section 4 an example of the application of the three algorithms is presented. In Section 5 computational experiments on randomly generated networks 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 $G(V, A, T_k)$, where G is a directed graph with a set of n nodes $V = \{1, \dots, n\}$, a set of m arcs $A \subseteq \{(j,l) : j,l \in V \text{ and } j \neq l\}$ and a finite set of acceptable parameters $T_k = \{t_i : i \in I_k\}$, with $I_k := \{1, \dots, k\}$, $k > 1$. Given the set T_k, a scenario $i \in I_k$ is determined according to the costs assigned under t_i. For each arc $(j,l) \in A$, j and l are named the tail and the head node, respectively. The associate cost function is defined by $c^k_{jl} : T_k \longrightarrow \mathbb{R}^+_0$, where $c^{i,k}_{jl} := c^k_{jl}(t_i)$ represents the cost of arc (j,l) in scenario i, or under parameter t_i. 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 $v_s \in V$, for $s = 2, \dots, r-1$, and $(v_s, v_{s+1}) \in A$, for $s = 1, \dots, r-1$. 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 i, or under t_i, $i \in I_k$, is defined by

$$v(p, t_i) = \sum_{(j,l) \in A(p)} c^{i,k}_{jl}. \tag{1}$$

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 there are no arcs arriving at 1 and no arcs starting at n in G. The set of all $(1,n)$-paths in G is represented by P.

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

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

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

$$RC(p) := \max_{i \in I_k} RD(p, t_i), \tag{3}$$

where $RD(p, t_i)$ represents the robust deviation of a path p under parameter t_i, $i \in I_k$. Let p^i_j be used to represent the j-th shortest $(1,n)$-path under t_i, the value $RD(p, t_i)$ is defined as

$$RD(p, t_i) := v(p, t_i) - v(p^i_1, t_i). \tag{4}$$

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

The set of scenarios in which $RC(p)$ occurs corresponds to the set of the indices of the parameters under which the robust deviation of $p \in P$ is maximized and it will be denoted by $I(p) := \{\arg\max_{i \in I_k} RD(p, t_i)\}$.

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 the minimax regret robust shortest path problem is the minmax shortest path problem [18]. This latter problem corresponds to an absolute version of problem (2), for which the robust deviation of a path p is replaced by its cost for the scenario in question. That is, the objective function to minimize is $\max_{i\in I_k}\{v(p,t_i)\}$. 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 $v(p_1^i,t_i)$ is a constant for any $i \in I_k$. In such case, the optimum values of the two problems differ by that constant.

2.1 Properties of the optimum 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 under t_1 and under t_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, under t_2 and under t_1, respectively ($I(p_1^1) = \{2\}$ and $I(q) = \{1\}$). Therefore, they are both robust shortest paths.

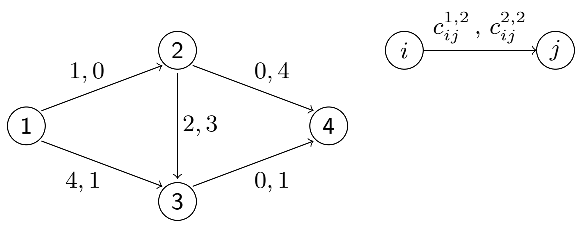


Figure 1: Network example

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

Lemma 1. *For every $p \in P$, $RC(p) \geq 0$. Moreover, $RC(p) = 0$ occurs if and only if p is a shortest path under every t_i, $i \in I_k$.*

Taking into account (2) and Lemma 1, 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 $i \in I_k$ then p is a robust shortest path, with $I(p) = I_k$.*

In order to develop algorithms that compute a path with the minimum robustness cost at a given network under a finite multi-scenario model, other properties must be established. An important result concerns the cyclic nature of an optimum 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 by the example 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 $q = \langle 1,2,3,4\rangle$, but a new robust shortest $(1,4)$-path $\langle 1,2,1,2,4\rangle$, containing the loop $C = \langle 1,2,1\rangle$, is obtained, given that it has the same minimum robustness cost in all scenarios.

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, stated in Lemma 3, is still valid for networks without cycles with negative cost in any scenario.

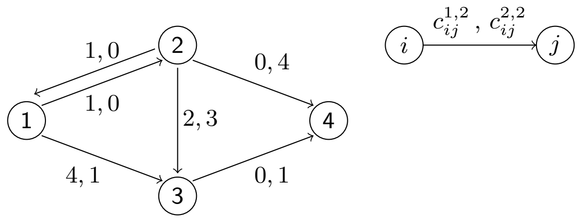


Figure 2: Network example with a cycle

Lemma 3. *Let G be a network with no cycles with negative cost in any scenario, then there exists a loopless optimum 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 optimum solution of the robust shortest path problem, then $v(q, t_i) \leq v(p, t_i)$, for any scenario $i \in I(p)$.*

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

3 Algorithms

The results established at Subsection 2.1 are a motivation to the development of three algorithms for solving (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 under a fixed scenario $i \in I_k$, 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 in network G 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 optimum path may contain sub-paths that are not optimum. Therefore Murthy and Her consider each scenario as one criteria and develop a labeling algorithm together 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 to that algorithm is the adaptation of the upper bounds for the values of the current objective functions. First, some concepts and notation are introduced.

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

$$z_1 = (-v(p_1^1, t_1), \dots, -v(p_1^k, t_k)).$$

Each component z_j^i is related with the cost of the $(1, j)$-path associated with label z_j, $j \in V$, in scenario $i \in I_k$. Moreover, along the algorithm, when such label is selected, all the new labels z_l, $(j, l) \in A$, can be created according to the formula

$$z_l = (z_j^1 + c_{jl}^{1,k}, \dots, z_j^k + c_{jl}^{k,k}). \tag{5}$$

With the initialization above, for $j = n$, $z_n = (RD(p_n, t_1), \dots, RD(p_n, t_k))$ is the vector of robust deviations of a given $(1, n)$-path, p_n. Hence, the solution space is explored through the labels for node n and the optimum value is obtained by selecting the label that exhibits 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. Then the $(1, n)$-path associated with z_n is a robust shortest path if and only if*

$$\max_{i \in I_k}\{z_n^i\} \leq \max_{i \in I_k}\{z_n'^i\} \tag{6}$$

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 $(1, j)$-paths associated with all existent labels z_j, $j \in V \backslash \{n\}$, to all possible paths with destination node n can be a heavy computational task. 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 optimum 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_j' and z_j'' be two labels associated with node j. Then z_j' dominates z_j'' if*

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

and at least one of the inequalities is strict.

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

Checking the dominance between different labels of a node is crucial to decide the ones that can be discarded. In fact, all labels z_j, $j \in V \backslash \{n\}$, dominated by another associated with the same node can be eliminated. Proposition 2 shows that any extension of the associate $(1, j)$-paths produces $(1, n)$-paths with a robustness cost that is never smaller that the robustness cost of paths associated with non-dominated labels. The storage of repeated labels of a node can be avoided, since robust shortest $(1, n)$-paths containing their associate partial paths have the same robustness cost.

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

$$\max_{i \in I_k}\{z_n^i\} \leq \max_{i \in I_k}\{z_n'^i\}.$$

Proof. Let p_{jn} denote the (j, n)-path that extends the associate $(1, j)$-paths with labels z_j and z'_j into $(1, n)$-paths with labels z_n and z'_n, respectively.

Since z_j dominates z'_j, one has $z_j^i \leq z_j'^i$ for every $i \in I_k$, with $z_j^s < z_j'^s$, for some $s \in I_k$, according to Definition 1. Then, $z_n^i = z_j^i + v(p_{jn}, t_i) \leq z_j'^i + v(p_{jn}, t_i) = z_n'^i$, for every $i \in I_k$, with $z_n^s < z_n'^s$. Consequently, $\max_{i \in I_k}\{z_n^i\} \leq \max_{i \in I_k}\{z_n'^i\}$. □

With a similar reasoning it can be shown 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 optimum loopless path exists that is formed only by nodes associated with the earliest possible label, when there are several 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_j^*, 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'_j. Denote by p_{jn}^* the (j, n)-path in p^*. Then $p'_j \diamond p_{jn}^*$ 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 $p'_j \diamond p_{jn}^*$, and $p'_x \diamond p_{xn}^*$ be the loopless $(1, n)$-path obtained from $p'_j \diamond p_{jn}^*$ after removal of that loop. Here p'_x and p_{xn}^* correspond to p_j's sub-path from 1 to x and p_{jn}^*'s sub path from x to n, respectively. Then, $p'_x \diamond p_{xn}^*$ and $p'_j \diamond p_{jn}^*$ 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'_x the result is proven. Otherwise the reasoning can be repeated. Because node labels are managed as a FIFO list, p'_x has less nodes than p_j^*, and therefore repeating it a finite number of times leads to a contradiction, as expected. □

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 tests are a pruning strategy adopted in [18], for all labels associated with any node $j \in V$. 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. Computational effort can, thus, be spared without affecting the determination of a loopless optimum solution.

A second pruning rule for the labels z_j, $j \in V \backslash \{n\}$, is inferred in Proposition 4. This property is based on a bounding condition verified by every sub-path of a robust shortest $(1,n)$-path. Assume that $LB_j = (LB_j^1, \ldots, LB_j^k)$ is a cost vector associated with a node $j \in V$, where the component LB_j^i represents the cost of the shortest (j,n)-path in scenario $i \in I_k$.

Proposition 4. *Let p be a robust shortest $(1,n)$-path and p_j be a $(1,j)$-sub-path of p with label z_j, $j \in V$. Then,*

$$\max_{i \in I_k} \{z_j^i + LB_j^i\} \leq RC(p). \tag{7}$$

Proof. Let p_j be a $(1,j)$-path, $j \in V \backslash \{n\}$, contained in a robust shortest $(1,n)$-path p, and let z_j be its label. As LB_j^i is a lower bound for the cost of any (j,n)-path under scenario t_i, then $v(p,t_i) \geq v(p_j,t_i) + LB_j^i$, or, equivalently, $RD(p,t_i) \geq z_j^i + LB_j^i$, for every $i \in I_k$, and condition (7) is satisfied. □

The second test for the labels z_j, $j \in V \backslash \{n\}$, allows to eliminate those that would produce complete $(1,n)$-paths with robustness costs greater than or equal to the least achieved. In fact, denoting by UB an upper bound for the optimum value of the problem, when condition

$$\max_{i \in I_k} \{z_j^i + LB_j^i\} \geq UB \tag{8}$$

holds, then the $(1,j)$-path associated with label z_j cannot be part of any optimum solution if the inequality is strict or it can be part of an optimum solution with robustness cost UB in case of equality. Nevertheless, taking into account that a candidate path with the same robustness cost UB has been previously selected, z_j can be discarded in both situations. 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, given that the calculation of their costs is fundamental to start the algorithm. Hence, the first value of UB is given by

$$\min_{i \in I_k} RC(p_1^i) = \min_{i \in I_k} \max_{j \in I_k} RD(p_1^i, t_j) = \min_{i \in I_k} \max_{j \in I_k} \{v(p_1^i, t_j) - LB_1^j\}. \tag{9}$$

Throughout the algorithm, UB is updated as new labels z_n are computed.

The main results that support the labeling algorithm for finding a robust shortest $(1,n)$-path have been established, its structure is described in the following.

Global algorithmic structure To start with, the computation of the trees $\mathcal{T}^i$ of shortest (j,n)-paths and of the associate cost lower bounds LB_j^i are necessary, for each scenario $i \in I_k$, $j \in V$. Any shortest path tree algorithm can be applied with such goal [1]. Then, the initialization of the cost upper bound UB for the optimum value of the problem is obtained, by (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, their robustness costs computation can be avoided by using a list Q with the distinct shortest paths over the scenarios of the network. In this way, one can adopt as the first candidate the path of Q with the least robustness cost.

A variable $RCaux$ is used to store the robustness cost of a $(1,n)$-path associated with a label z_n under analysis and *sol* represents the corresponding path, which is an optimum path candidate.

Another list X is used with the purpose of storing all the labels z_j to be scanned, $j \in V \backslash \{n\}$. The scanned labels of node j which are not eliminated are stored in a list Z_j and the associate predecessor nodes are inserted in a list P_j. This list is used in order to retrieve the optimum solution by tracing back the nodes up to node 1.

The labeling algorithm for finding a robust shortest $(1, n)$-path is organized as follows:

- **Initialization**: Every tree $\mathcal{T}^i$ of (j, n)-shortest paths, $i \in I_k$, $j \in V$, is determined by a shortest path algorithm. The obtained shortest $(1, n)$-path p_1^i is inserted in list Q, if not yet in Q, and the costs LB_j^i are set according to the shortest path algorithm output.

  The initial candidate solution *sol* is set to the path with the least robustness cost in Q. This value is assigned to the initial UB.

  Afterwards, the initial label is $z_1 = (-LB_1^1, \dots, -LB_1^k)$. This label is inserted in list X and the lists Z_j and P_j, $j \in V \backslash \{1, n\}$, are initialized as empty lists, whereas Z_1 is set to $\{z_1\}$.

- **Iterations**: While there exist labels to scan in list X, the first one is extracted, z_j, and the labels z_l, for any $(j, l) \in A$, are created according to (5). Two situations may arise:

  1. If $l = n$, the robustness cost of the $(1, n)$-path associated with label z_n is calculated by $\max_{i \in I_k}\{z_n^i\}$. This value updates UB if it is smaller than the previous value and *sol* is updated with the current $(1, n)$-path. This path can be retrieved by tracing back all the predecessors from n's predecessor node, j, starting with list P_j, and adding arc (j, n) at the end.
  2. If $l \neq n$, the dominance test of label z_l is performed among the remaining labels stored in Z_l. If none of the elements in Z_l is equal to z_l or dominates it and condition (8) does not hold, then the $(1, l)$-path associated with label z_l is considered for further extension, as it may be part of a potential optimum $(1, n)$-path with a robustness cost smaller than the value UB, according to Propositions 2 and 4. If such conditions are satisfied the elements of X and Z_l are compared with z_l, in order to eliminate any dominated labels. Finally, z_l is inserted in lists X and Z_l for further evaluation, whereas l's predecessor node is included in list P_l.

- **Return**: *sol* (robust shortest $(1, n)$-path).

The pseudo-code of the procedure outlined above is presented in Algorithm 1.

Computational time complexity order In order to determine the computational complexity of the algorithm when considering a worst case, the time upper bound for some auxiliary procedures must be analyzed.

1. **Determination of a tree $\mathcal{T}^i$ of shortest (j, n)-paths, $j \in V$, in a scenario $i \in I_k$, and 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^i, $i \in I_k$, given $LB_1^1, \dots, LB_1^k$**: For the calculation of the robustness cost of a path p_1^i, all the costs $v(p_1^i, t_j)$, $j \in I_k$, were determined in 1. The k robust deviations, $RD(p_1^i, t_j)$, $j \in I_k$, 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 $\mathcal{O}(k)$.

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

1 $Q \leftarrow \emptyset$;
2 **for** $r = 1, \ldots, k$ **do**
3 Compute the tree $\mathcal{T}^r$ under t_r;
4 $Q \leftarrow Q \cup \{p_1^r\}$;
5 **for** $j = 1, \ldots, n$ **do** $LB_j^r \leftarrow$ cost of the shortest (j, n)-path under t_r;
6 ;
7 $UB \leftarrow \min\{RC(p_1^r) : r \in I_k\}$;
8 $sol \leftarrow p_1^r$ such that $RC(p_1^r) = UB$, $r \in I_k$;
9 $z_1 \leftarrow (-LB_1^1, \ldots, -LB_1^k)$;
10 $X \leftarrow \{z_1\}$; $Z_1 \leftarrow \{z_1\}$;
11 **for** $j = 2, \ldots, n-1$ **do** $Z_j \leftarrow \emptyset$; $P_j \leftarrow \emptyset$;
12 ;
13 **while** $X \neq \emptyset$ **do**
14 $z_j \leftarrow$ first label in X; $X \leftarrow X - \{z_j\}$;
15 **for** $(j, l) \in A$ **do**
16 $z_l \leftarrow (z_j^1 + c_{jl}^{1,k}, \ldots, z_j^k + c_{jl}^{k,k})$;
17 **if** $l = n$ **then**
18 $RCaux \leftarrow \max\{z_l^i : i \in I_k\}$;
19 **if** $RCaux < UB$ **then**
20 $UB \leftarrow RCaux$;
21 $p_j \leftarrow (1, j)$-path constructed back to node 1 by starting with P_j;
22 $sol \leftarrow p_j \diamond \langle j, n\rangle$;
23 **else if** z_l *is not dominated by or equal to any label in* Z_l **and** $\max_{i \in I_k}\{z_l^i + LB_l^i\} < UB$ **then**
24 Delete from X and from Z_l all the labels of l that are dominated by z_l;
25 Delete from P_l the predecessor nodes associated with the labels deleted from Z_l;
26 $X \leftarrow X \cup \{z_l\}$; $Z_l \leftarrow Z_l \cup \{z_l\}$; $P_l \leftarrow P_l \cup \{j\}$;
27 **return** sol

3. **Calculation of a label z_l from a given label z_j, with** $(j, l) \in A$: According to (5), label z_l is determined from z_j by adding the k costs of arc (j, l), hence it is obtained in $\mathcal{O}(k)$ time.

4. **Dominance test between two given labels**: Since in a worst case all the components of two given labels are considered on a dominance test, at most k comparisons are involved and consequently this operation has $\mathcal{O}(k)$ time.

Algorithm 1 is performed in two stages. The first one consists of the initialization steps for the computation of the trees $\mathcal{T}^i$ and includes the calculation of the costs of their shortest (j, n)-paths, $j \in V$. This is done in $\mathcal{O}(km + k^2 n)$ for acyclic networks and in $\mathcal{O}(k(m + n \log n) + k^2 n)$ for general networks, according to 1. In a worst case all the shortest paths $p_1^1, \ldots, p_1^k$ belong to list Q and the calculation of their k robustness costs is done in $\mathcal{O}(k^2)$ time, attending to 2. Initializing the upper bound UB by selecting the minimum robustness cost requires $\mathcal{O}(k)$ time. Hence, the total amount of operations that precede the generation of the labels is performed in $O_1^a = \mathcal{O}(km + k^2 n)$ time for acyclic networks and $O_1^c = \mathcal{O}(k(m + n \log n) + k^2 n)$ 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, by 3. the calculation of a new label is done in $\mathcal{O}(k)$ time, by 4. the dominance tests are performed in $\mathcal{O}(kW)$, and (8) is checked in $\mathcal{O}(1)$. Moreover, updating X, Z_j, P_j, $j \in V$, 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 $\log n \ll n$.

3.2 Ranking algorithms

An alternative strategy for determining a loopless robust shortest path based on the loopless paths ranking combined with the definition of a cost upper bound is derived in the following. This technique is inspired on the work of Dias and ClÃmaco [6] to find bicriteria shortest paths and provides a method supported on ranking loopless paths under a fixed scenario in which the optimum path(s) can be determined by fixing a given cost upper-bound. Dias and ClÃmaco used the algorithm by ClÃmaco and Martins [5], where the cost upper bound is chosen in such a way that all paths that cannot be dominated in terms of cost by any other over all scenarios stay at the delimited region. This strategy was equally useful on continuous models for the same problem, after the discretization of the costs interval data by taking their lower and upper bounds. For the robust shortest problem with finite multi-scenarios, the existence of an optimum solution, even if not unique, is assured. Hence, some adaptations to the previous method can be made, taking into account the computation of a single loopless optimum solution and the new optimum values according to the number of scenarios involved. Consequently, the adoption of a loopless paths ranking is required and the update of the costs upper bounds according to the least produced optimum values can be done till an optimum path is found. Two versions based on such method will be presented. The first is a procedure that ranks loopless paths and sets basic upper bounds for the costs under a given scenario. The second is an alternative version of the former, enhanced by the application of reduction techniques to the ranking similar to those presented for the labeling approach.

3.2.1 Basic version

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 cost under a fixed scenario $i \in I_k$. Also, a stopping criterion is imposed by means of a particular cost upper bound. On the one hand, Lemma 3 shows that ranking unconstrained paths can be avoided, even in networks with cycles. On the other hand, Proposition 1 allows to establish an upper limit for the ranking, associated with the determined candidates to robust shortest path. Specifically, once a candidate loopless path p is returned by the ranking in scenario i, the search for other candidates consists of ranking loopless paths with a cost smaller than $v(p, t_i)$. 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 optimum solution. Since $i \in I(p)$, $RD(p, t_i) = RC(p)$ and, consequently, $v(p, t_i)$ can be rewritten as $LB_1^i + RC(p)$. When this cost upper bound is set for any robust shortest path candidate p, it can be reduced whenever a path q satisfying $RC(q) < RC(p)$ is found along the ranking. The next result shows this and that it is even possible to detect an exact solution when $i \in I(q)$.

Proposition 5. *Let $p \in P$ and any $i \in I_k$. Let $q \in P \backslash \{p\}$ verify:*

1. $v(q, t_i) \leq LB_1^i + RC(p)$,

2. $RC(q) < RC(p)$,

3. $RC(\tilde{q}) \geq RC(p)$, $\forall \tilde{q} \in P \backslash \{q\} : v(\tilde{q}, t_i) < v(q, t_i)$.

Then, any solution $\tilde{p}$ of problem (2) satisfies

$$v(q, t_i) \leq v(\tilde{p}, t_i) \leq LB_1^i + RC(q) < LB_1^i + RC(p). \tag{10}$$

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

Proof. Let $p \in P$, $i \in I_k$ and $q \in P \backslash \{p\}$, such that q satisfies conditions 1., 2. and 3.

By conditions 1. and 2., one has $LB_1^i + RC(q) < LB_1^i + RC(p)$. Besides, condition 2. and definition (2) allow to conclude that a robust shortest path must have a robustness cost not greater than $RC(q)$. Therefore, the robust deviation of every optimum solution $\tilde{p}$ of (2) under t_i must satisfy $RD(\tilde{p}, t_i) \leq RC(q)$, which implies

$$\forall \tilde{p} \in P : \tilde{p} \text{ is solution of (2)}, \ v(\tilde{p}, t_i) \leq RC(q) + LB_1^i < LB_1^i + RC(p). \tag{11}$$

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

$$\forall \tilde{p} \in P : \tilde{p} \text{ is solution of (2)}, \ v(\tilde{p}, t_i) \geq v(q, t_i). \tag{12}$$

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

In addition to 1., 2. and 3., let now be assumed that $i \in I(q)$. For this case, t_i is the parameter under which the robust deviation of q is maximized, which means that $RC(q) = RD(q, t_i)$. Thus, (10) becomes an equality, since $RC(q) + LB_1^i = RD(q, t_i) + LB_1^i = v(q, t_i)$, i.e.,

$$\forall \tilde{p} \in P : \tilde{p} \text{ is solution of (2)}, \ v(\tilde{p}, t_i) = v(q, t_i). \tag{13}$$

On the other hand,

$$\forall \tilde{q} \in P \backslash \{q\} : v(\tilde{q}, t_i) = v(q, t_i), \ RC(\tilde{q}) \geq RD(\tilde{q}, t_i) = RD(q, t_i),$$

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

$$\forall \tilde{q} \in P \backslash \{q\} : v(\tilde{q}, t_i) = v(q, t_i), \ RC(\tilde{q}) \geq RC(q).$$

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

This result gives two necessary conditions concerning bounding the robust shortest path cost for the ranking scenario $i \in I_k$ and detecting a robust shortest path under specific assumptions. When considering the cost upper bound $LB_1^i + RC(p)$ for a loopless path p, if a loopless path q is found such that $RC(q) < RC(p)$, then attending to (10) a new upper bound can be set, $LB_1^i + RC(q)$. This decreases the cost upper bound, and thus shortens the ranking. Moreover, the analysis of the scenarios in which $RC(q)$ occurs, i.e. of the indices of the parameters for which the robust deviation of q is maximized, and their identification with i is crucial to spare computational effort by allowing the search to halt when $i \in I(q)$, because in this case it can be concluded that q is an optimum solution.

The efficiency of the method here proposed is related with the coincidence of the scenario in which the minimum robustness cost occurs with the scenario i in which the ranking is performed and the fact that a solution is relatively close to p_1^i, in order to stop the algorithm as early as possible.

Analogously to Algorithm 1, the cost upper bound for the optimum value of problem (2) 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 $i \in I_k$, in which the ranking is performed. Without loss of generality, it will be chosen the scenario with the smallest index i for which the robustness cost of the first candidate optimum solution p^*, with $p^* = p_1^j$, for some $j \in I_k$, occurs, i.e.

$$i = \min \{ r \in I_k : RD(p^*, t_r) = RC(p^*) \}. \tag{14}$$

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

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

Global algorithmic structure The preliminary procedures of this approach, namely the initialization of the ranking cost upper bound, are similar to Algorithm 1. A list Q of distinct shortest paths for all scenarios is used. Analogously, the variables $RCaux$, UB and sol represent the robustness cost of the current path, the best robustness cost and the best candidate loopless path found so far, respectively. Another variable stores the cost upper bound for the ranking, $Vmax$.

For general networks several algorithms can be applied to rank loopless paths under t_i, 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^i, $i \in I_k$, that was already analyzed, allowing to avoid its robustness cost recalculation.

The algorithm for finding a robust shortest $(1, n)$-path is organized as follows.

- **Initialization**: The paths p_1^i, $i \in I_k$, are determined by a shortest path algorithm and list Q stores them with no repetitions.

  The initial candidate solution sol is set to the path in Q with the least robustness cost, this value is assigned to the initial UB.

  As mentioned above, the first index i for which $RD(sol, t_i) = UB$ determines the scenario in which the ranking will be performed.

  Afterward, $Vmax$ is set to $v(sol, t_i)$, the cost upper bound for the ranking, and variable j, responsible for identifying the index of the current path in the ranking, is assigned to 2.

- **Iterations**: While the j-th shortest loopless path under t_i, p_j^i, exists, if $v(p_j^i, t_i) \geq Vmax$, one halts the ranking; otherwise, in case p_j^i does not belong to Q, then it was not scanned yet, which means that its robustness cost, $RCaux$, must be determined and compared with UB.

  Afterward, if $RCaux < UB$, then UB and sol must be updated with $RCaux$ and p_j^i, respectively. According to Proposition 5, in case $i \in I(p_j^i)$ (i.e. $RD(p_j^i, t_i) = UB$), then p_j^i is an optimum solution and the computation can halt. Otherwise, by (10), a new cost upper bound is fixed to $LB_1^i + UB$.

  The next step is to increment j and then the algorithm continues to the next iteration.

- **Return**: sol (robust shortest $(1, n)$-path).

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

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

---

$Q \leftarrow \emptyset$;
for $r = 1, \ldots, k$ **do**
    $p_1^r \leftarrow$ shortest path under t_r;
    $Q \leftarrow Q \cup \{p_1^r\}$;
$UB \leftarrow \min\{RC(p_1^r) : r \in I_k\}$;
$sol \leftarrow p_1^r$ such that $RC(p_1^r) = UB$, $r \in I_k$;
$i \leftarrow \min \big\{r \in I_k : RD(sol, t_r) = UB\big\}$;
$Vmax \leftarrow v(sol, t_i)$;
$j \leftarrow 2$;
while *the j-th shortest loopless path under t_i, p_j^i, exists* **do**
    Compute p_j^i;
    if $v(p_j^i, t_i) \geq Vmax$ **then** **break**;
    ;
    else
        if $p_j^i \notin Q$ **then**
            $RCaux \leftarrow RC(p_j^i)$;
            if $RCaux < UB$ **then**
                $UB \leftarrow RCaux$;
                $sol \leftarrow p_j^i$;
                if $RD(p_j^i, t_i) = UB$ **then** **break**;
                ;
                else $Vmax \leftarrow v(p_1^i, t_i) + UB$;
                ;
    $j \leftarrow j + 1$;
return sol

Computational time complexity order In order to determine the worst case time computational complexity of the algorithm, the upper bounds of the involved procedures will be analyzed.

Algorithm 2 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 will be used regarding the complexity of the first phase of Algorithm 2, given that the computation of list Q and the selection of the first candidate solution with the least robustness cost are analogous. With the same reasoning such procedures are performed in $O_1^a = \mathcal{O}(km + k^2 n)$ time for acyclic networks and in $O_1^c = \mathcal{O}(k(m + n \log n) + k^2 n)$ time for general networks. According to (14), the choice of the scenario in which the ranking will be defined 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 i after the shortest path being computed, 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, other than p_1^i, have to be determined. The cost of each of those paths under 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 $O_1^a + O_2^a = \mathcal{O}(k^2 n + k\max\{m, Ln\} + L\log L)$ for acyclic networks and $O_1^c + O_2^c = \mathcal{O}(k^2 n + \max\{k, Ln\}(m + n\log n) + kLn)$ for general networks. Both bounds are pseudo-polynomial, in the sense that L depends on the input model parameters.

3.2.2 Hybrid version

The method presented in this section to determine a robust shortest path 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 and the rules used to discard useless paths are presented. Unless otherwise stated, it is assumed that the ranking is done with respect to a given scenario $i \in I_k$.

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 r-shortest path candidates, $r > 1$, 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, the new candidates should have the form

$$q_{w,x}^{p,i} = p_{1w} \diamond \langle w, x\rangle \diamond p_x^{*i}\,, \quad x \in V^+(w), \tag{15}$$

where p_x^{*i} represents the shortest (x,n)-path for scenario $i \in I_k$ in the tree $\mathcal{T}^i$, for any $x \in V$, and $V^+(w) = \{x \in V : (w,x) \in A\}$ is the set of head nodes of the arcs in G with tail node w. In this case p is called the father of $q_{w,x}^{p,i}$. Additionally, w and (w,x) are denominated the deviation node and the deviation arc of path $q_{w,x}^{p,i}$, 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 i 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, MPS algorithm replaces the arc costs by reduced costs, as explained next. Recalling that LB_j^i denotes the cost of the shortest (j,n)-path in scenario i, $j \in V$, the reduced cost $\bar{c}_{sl}^{i,k}$ of an arc $(s,l) \in A$ associated with i is defined by

$$\bar{c}_{sl}^{i,k} = LB_l^i - LB_s^i + c_{sl}^{i,k}.$$

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

$$\bar{v}(p, t_i) = \sum_{(j,l)\in A(p)} \bar{c}_{jl}^{i,k}. \tag{16}$$

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

Let $\widehat{V}^{+i}(w) = \{x_1, \ldots, x_{l_w}\}$ represent the set $V^+(w)$ sorted by non-decreasing order of the reduced costs of the associate arcs with respect to scenario i, that is, such that $\bar{c}_{wx_1}^{i,k} \leq \ldots \leq \bar{c}_{wx_{l_w}}^{i,k}$. Therefore,

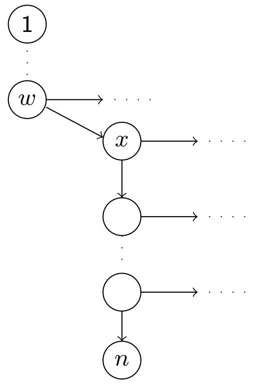
$\bar{v}(q^{p,i}_{w,x_1}, t_i) \leq \ldots \leq \bar{v}(q^{p,i}_{w,x_{l_w}}, t_i)$ and, thus, the costs in scenario i of the paths generated from a $(1, n)$-path p by deviation at node $w \in V(p)$ are sorted in the following way:

$$v(q^{p,i}_{w,x_1}, t_i) \leq \ldots \leq v(q^{p,i}_{w,x_{l_w}}, t_i). \tag{17}$$

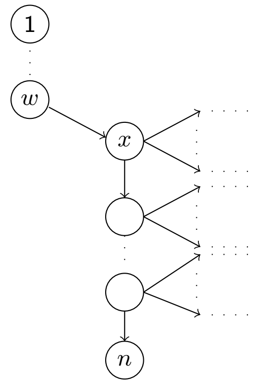
Assuming that p_{1w} is a loopless $(1, w)$-path, the deviation path $q^{p,i}_{w,x}$ results from the concatenation of three loopless paths and therefore it can still contain repeated nodes. However, by comparing the possible nodes x with the nodes in p_{1w} the choice of x can be done in such a way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

$$V^{+i}(p_{1w}, x_u) = \{x_j \in \widehat{V}^{+i}(w) : j > u \text{ and } p_{1w} \diamond \langle w, x_j\rangle \text{ is loopless } \}. \tag{18}$$

This set contains the head nodes of the deviation arcs associated with the path p_{1w} with a reduced cost under t_i of at least $\bar{c}^{i,k}_{wx_u}$ and such that $p_{1w} \diamond \langle w, x_j\rangle$ is still loopless. The nodes considered for each path p are those from p's deviation node to the node that precedes n. A scheme of the deviation paths with respect to a path with deviation arc (w, x) generated by MPS algorithm is shown in Figure 3.(a).



(a) MPS version



(b) Hybrid version

Figure 3: Deviation techniques used in MPS and hybrid algorithms

In the MPS algorithm the deviation from path p at node w is obtained by taking the first head node x_j in the ordered set $V^{+i}(p_{1w}, x_u)$. 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 $\widehat{V}^{+i}(w)$ is sorted according to non-decreasing order the reduced costs, for any $w \in V$ [14]. For scenario i, MPS algorithm starts to generate deviations from the shortest path p^i_1 at every of its nodes but n. The resulting paths, one per each scanned node, are stored in a list and are selected, by order non-decreasing of the reduced costs, in future iterations. Each of these paths is identified as the r-th loopless shortest path with respect to scenario i in case it is loopless, for some $r > 1$. This process is performed iteratively under the same conditions. Scanning only the nodes of its 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 latter method, at most one new deviation path is generated when scanning a given 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. If looking for robust shortest paths using a ranking approach, the more solution candidates are generated the higher the chances of computing paths with smaller robustness costs in an early stage. An expected consequence is to reduce faster the cost upper bound under t_i, and possibly to find an optimum solution quicker. Thus, a technique similar to the generalization of Yen's algorithm described in [14] is used. 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) restricted to the arcs in the set $V^{+i}(p_{1w}, x_u)$, in (18), and chosen according to the 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 optimum solution. Figure 3.(b) illustrates the deviation technique with respect to a path with deviation arc (w, x) used in the hybrid algorithms.

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

Corollary 1. *Let $p \in P$, $w \in V(p)\backslash\{n\}$, $x \in V^{+i}(w)$, and let $q^{p,i}_{w,x} = p_{1w} \diamond \langle w, x\rangle \diamond p^{*i}_x$ 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. $\max_{r\in I_k}\{v(p_{1w}, t_r) + LB^r_w - LB^r_1\} \leq RC(\tilde{p})$;
2. $v(\tilde{p}, t_i) \leq v(q^{p,i}_{w,x}, t_i)$, *for every* $i \in I(q^{p,i}_{w,x})$.

The first point of this corollary is a sufficient condition for a deviation path of a $(1, n)$-path to produce a candidate to an optimum 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 robust shortest path which is a deviation path.

In the following the bounds given by Corollary 1 are enhanced taking into account the results used in Algorithm 2. Let $Vmax$ and UB denote upper bounds for the paths cost in scenario i 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^j_1, $j \in I_k$. For a $(1, n)$-path p and a node $w \in V(p)\backslash\{n\}$, such that $(w, x_u) \in A(p)$ and $u < l_w$, every arc (w, x'_u), with $x'_u \in V^{+i}(p_{1w}, x_u)$, is considered as a new deviation arc. Assuming that the set $V^{+i}(p_{1w}, x_u) = \{x_{u'_1}, \ldots, x_{u'_s}\}$ is sorted, the deviation paths $q^{p,i}_{w,x_{u'_1}}, \ldots, q^{p,i}_{w,x_{u'_s}}$ 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 $\max_{r\in I_k}\{v(p_{1w}, t_r) + LB^r_w - LB^r_1\} > UB$. In this case, all the deviation paths $q^{p,i}_{w,x}$, with $x \in V^{+i}(p_{1w}, x_u)$, can be skipped.
2. The rule above can be refined given that, by the same result, if $\max_{r\in I_k}\{v(p_{1w}, t_r) + c^{r,k}_{wx'_j} + LB^r_{x'_j} - LB^r_1\} > UB$, the path $p_{1w} \diamond \langle w, x'_j\rangle \diamond p^{*i}_{x'_j}$ and its subsequent deviations will not lead to optimum paths, and thus can be skipped.
3. Let j be the first element in $\{1, \ldots, s\}$ such that $v(q^{p,i}_{w,x_{u'_j}}, t_i) > Vmax$. Then, all the deviation paths $q^{p,i}_{w,x_{u'_{j'}}}$, $j \leq j' \leq s$, can be discarded.

4. Let $r \in \{1, \dots, s\}$ be the smallest index such that $RC(q^{p,i}_{w,x_{u'_r}}) = RD(q^{p,i}_{w,x_{u'_r}}, t_i) \leq UB$. By point 2. of Corollary 1 all paths of form $q^{p,i}_{w,x_{u'_{r'}}}$, with $r < r' \leq s$, and subsequent deviations, produce 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 to be considered, p^i_1, 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 of a path obtained from the deviation process is the tail node of an arc in $\mathcal{T}^i$. This is valid both for p^i_1, which is a path in such 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 $w \in p$ that is scanned and $(w, x) \in A(p)$, it holds $\bar{c}^{i,k}_{wx} = 0$. Therefore, x is the first element in $\widehat{V}^{+i}(w)$, i.e. $x = x_1$, and the available head nodes of the deviation arcs with tail node w stay in $V^{+i}(p_{1w}, x_1)$. The obtained candidate paths are stored in a list X and the path with the least cost under t_i is chosen to be deviated in the next iteration. Throughout the algorithm the upper bounds $Vmax$ and UB are updated and a loopless robust shortest path is identified when the stopping criterion used in Algorithm 2 is met.

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 is created to store all 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 $Vmax$ have the same meaning as in Algorithm 2. The path being scanned is denoted 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. A scheme of the algorithm is now sketched.

- **Initialization**: The trees $\mathcal{T}^i$, $i \in I_k$, list Q, bounds LB^i_j, $j \in V$, and the initial candidate solution *sol* are determined like in Algorithm 1. The initial variables UB and $Vmax$, and the scenario i in which the ranking is performed are then determined like in Algorithm 2.

  The lists X and W are initialized as empty sets and p^i_1 is assigned with the first path p for analysis.

- **Iterations**: While there is a path p for being scanned such that $RD(p, t_i) \neq UB$, additional paths candidate to be optimum can be generated from each node $w \in V(p) \backslash \{n\}$ from the head node of p's deviation arc to the node that precedes either n or the first which is repeated in p. If $(w, x_1) \in A(p)$, then x_1 is the first element in $\widehat{V}^{+i}(w)$. Then, the arcs (w, x) are considered as deviation arcs by non-decreasing order of the reduced costs, for $x \in V^{+i}(p_{1w}, x_1)$. If $\max_{r \in I_k}\{v(p_{1w}, t_r) + c^{r,k}_{wx} + LB^r_x - LB^r_1\} \leq UB$, then (w, x) can lead to an optimum path and the deviation path $q^{p,i}_{w,x}$ is generated. If $v(q^{p,i}_{w,x}, t_i) > Vmax$, all paths to be computed after $q^{p,i}_{w,x}$ with deviation node w are skipped. Otherwise, $q^{p,i}_{w,x}$ is stored in list W and its robustness cost, $RCaux$ is determined and compared with UB. If $RCaux < UB$, UB is updated to $RCaux$ and the cost upper bound, $Vmax$, is updated to $LB^i_1 + UB$. After this, the deviation paths containing p_{1w} as sub-path in list W with cost under t_i greater than $Vmax$ or for which $\max_{r \in I_k}\{v(p_{1w}, t_r) + c^{r,k}_{wx} + LB^r_x - LB^r_1\} > UB$ are removed. Moreover, if some path $q^{p,i}_{w,x}$ satisfies $RD(q^{p,i}_{w,x}, t_i) = UB$, then all the forthcoming deviation paths with deviation node w are avoided.

  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 optimum path candidate, and *sol* is updated. Besides, the paths in X that satisfy the pruning rules above (points 2. or 3.) are removed from the list. Finally, all the paths stored in W are inserted in list X and the next path to be considered is the shortest under t_i in X.

- **Return**: *sol* (robust shortest $(1, n)$-path).

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

Computational time complexity order Algorithm 3 has two phases. Like for the previous approaches the first phase, related with the preliminary procedures, can be performed in $\mathcal{O}_1^a = \mathcal{O}(km + k^2 n)$ for acyclic networks and $\mathcal{O}_1^c = \mathcal{O}(k(m + n\log n) + k^2 n)$ 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 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 is performed once and 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 $\max_{r \in I_k}\{v(p_{1w}, t_r) + c_{wx}^{i,r} + LB_x^r - LB_1^r\} \leq UB$, which can be done in $\mathcal{O}(k)$ time. The third test involves the calculation of the cost of the deviation path $q_{w,x}^{p,i}$ under scenario i in $\mathcal{O}(1)$, by summing previously known values and checking $v(q_{w,x}^{p,i}, t_i) \leq Vmax$. The fourth test implies the calculation of $RC(q_{w,x}^{p,i})$, $\mathcal{O}(k)$, and its comparison with UB, $\mathcal{O}(1)$. 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 $O_1^a + O_2^a = \mathcal{O}(k^2 n + kmH + m \log n)$ for acyclic networks, and $O_1^c + O_2^c = \mathcal{O}(k^2 n + k \max\{mH, n \log n\} + m \log n)$ for general networks. Like parameter L used in Algorithm 2, H depends on n, m and k and cannot be known in advance. Moreover, $H \geq L$.

4 Example

Let $G(V, A, T_2)$ be the network depicted in Figure 4, and consider the application of Algorithms 1, 2 and 3 for finding a robust shortest $(1, 6)$-path in $G(V, A, T_2)$.

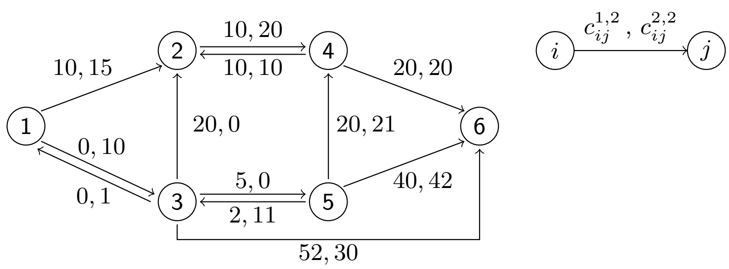
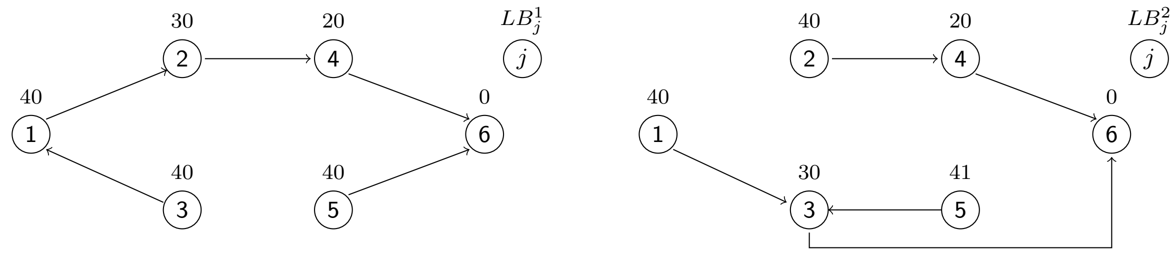


Figure 4: Network $G(V, A, T_2)$

The plots in Figure 5 show the tree of the shortest paths from every node to node 6 under scenario 1 – Figure 5.(a) – and the tree of the shortest paths from every node to node 6 under scenario 2 – Figure 5.(b). The values attached to a tree node j represent the cost of the $(j, 6)$-path in that tree. For this example, one has $LB_1 = (40, 40), LB_2 = (30, 40), LB_3 = (40, 30), LB_4 = (20, 20), LB_5 = (40, 41)$.



(a) under scenario 1 (b) under scenario 2

Figure 5: Shortest path trees rooted at $n = 6$ in $G(V, A, T_2)$

Initially, the elements of set Q are given by the shortest paths under parameters t_1 and t_2, i.e. $p_1^1 = \langle 1, 2, 4, 6\rangle$, with $v(p_1^1, t_1) = 40$, and $p_1^2 = \langle 1, 3, 6\rangle$, with $v(p_1^2, t_2) = 40$, respectively. Because $v(p_1^2, t_1) = 52 < v(p_1^1, t_2) = 55$, 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 optimum solution, $sol = p_1^2$. Since $I(p_1^2) = \{t_1\}$ the loopless paths will be ranked in scenario 1 for Algorithms 2 and 3, and $Vmax = v(p_1^2, t_1) = 52$ is considered as the initial cost upper bound.

Application of Algorithm 1

Figure 6 shows the tree of paths that is obtained by the application of Algorithm 1.

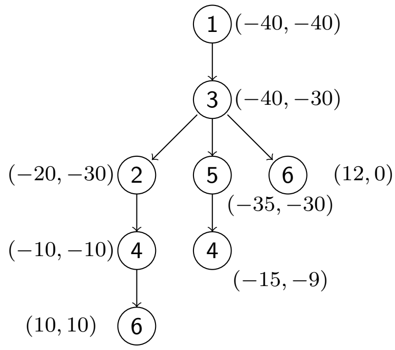


Figure 6: Search tree of paths produced by Algorithm 1

The method starts by selecting the label $z_1 = (-v(p_1^1, t_1), -v(p_1^2, t_2)) = (-40, -40)$ 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 $\max_{r\in I_2}\{z_2^r + LB_2^r\} = 15 > 12$ and thus it leads to paths with a robustness cost greater than 12. On the other hand, $\max_{r\in I_2}\{z_3^r + LB_3^r\} = 0 \leq 12$, which means that the extension of the associate $(1, 3)$-path might be optimum. 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 $\max_{r\in I_2}\{z_2^r + LB_2^r\} = 10 \leq 12$ and $\max_{r\in I_2}\{z_5^r + LB_5^r\} = 11 \leq 12$. The label z_6 satisfies $\max_{r\in I_2}\{z_6^r\} = 12$, 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, according to $\max_{r\in I_2}\{z_4^r + LB_4^r\} = 10 \leq 12$, 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 $\max_{r\in I_2}\{z_4^r + LB_4^r\} = 11 \leq 12$, so z_4 is inserted in Z_4. The two labels in Z_4, $(-10,-10)$ and $(-15,-9)$, produce labels of node 6, given by $(10,10)$ and $(5,11)$, respectively. The first of them has a least robustness cost of 10, which corresponds to the associate path $\langle 1,3,2,4,6\rangle$, the robust shortest $(1,6)$-path.

Application of Algorithm 2

Table 1 summarizes the steps of Algorithm 2 when applied to the network in Figure 4 till a robust shortest path is determined.

The computation of the second shortest path under t_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 $Vmax = v(p_3^1, t_1) = 51$ for the ranking is also set and the potential optimum solution sol is updated to p_3^1. Since the maximum robust deviation of p_3^1 does not occur under t_1, i.e. $RD(p_3^1, t_1) \neq 11$, 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 and under the ranking parameter t_1. This implies to halt the algorithm and to return p_4^1 as the optimum solution.

Application of Algorithm 3

The tree of the paths obtained with Algorithm 3 is depicted in Figure 7.

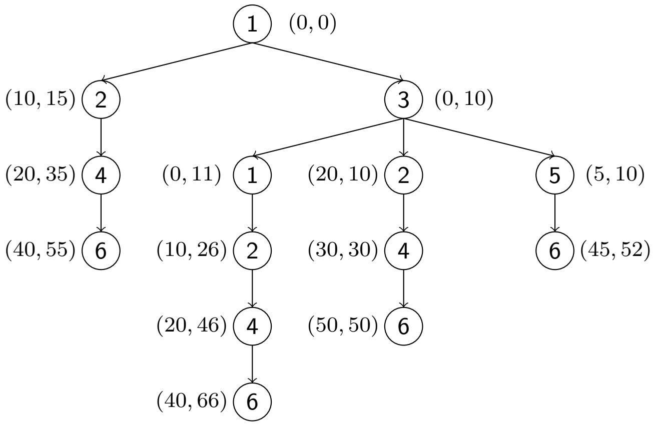


Figure 7: Tree of the deviation paths produced by Algorithm 3

The method starts by considering the same initial upper bounds as in Algorithm 2, $Vmax = 52$, $UB = 12$, and deviating from $p_1^1 = \langle 1,2,4,6\rangle$ at 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 $v(q_{1,2}^{p_1^1,1}, t_1) = 40 < 52$, which contains a loop. This is the path p considered in the following iteration, and this time only the node 3 is scanned. Besides, $\max_{r\in I_2}\left\{v(\langle 1,3\rangle, t_r) + LB_3^r - LB_1^r\right\} = 10 \leq 12$, thus potential optimum 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. By using deviation arc $(3,5)$, the loopless path $q^{p,1}_{3,5} = \langle 1,3,5,6\rangle$ is computed, with $v(q^{p,1}_{3,5}, t_1) = 45 < 52$ and robustness cost $RC(q^{p,1}_{3,5}) = 12$, 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 $q^{p,1}_{3,2} = \langle 1,3,2,4,6\rangle$ is obtained. This path satisfies $v(q^{p,1}_{3,2}, t_1) = 50 < 52$ and $RC(q^{p,1}_{3,2}) = 10 < 12$, so it is a new candidate for optimality and it is stored in W as well. Moreover, since $RD(q^{p,1}_{3,2}, t_1) = 10$, 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 $v(\langle 1,3,5,6\rangle, t_1) = 45 < 50$, path $\langle 1,3,5,6\rangle$ satisfies $\max_{r\in I_2}\{v(\langle 1,3\rangle, t_r) + c^{r,k}_{35} + LB^r_5 - LB^r_1\} = 11 > 10$. Thus, it will not produce optimum 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

In order to evaluate the performance of the introduced methods, Algorithms 1, 2 and 3 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. These implementations use Dijkstra's algorithm [1] to solve the shortest path problem for every scenario i and construct the associate tree $\mathcal{T}^i$. For Algorithm 1, a FIFO selection of the labels in list X is adopted. MPS algorithm [14] is applied to rank the loopless paths in Algorithm 2.

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 $U(0,100)$, 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 $n \in \{250,500,750\}$ and $d \in \{5,10,15\}$, where $d = m/n$ represents the density.

For each network dimension of each type, 10 problems were generated and solved by Algorithms 1, 2 and 3.

5.2 Results

From now on Algorithms 1, 2 and 3 will be represented by the abbreviations LA, RA and HA, respectively. In order to analyze their performances, their total running times are evaluated (in seconds).

For each algorithm, the total CPU is subdivided into two partial CPU times, the cpu_1 concerning the trees $\mathcal{T}^i$ computation, $i \in I_k$, at the initialization step and the cpu_2 required for the remaining procedures. The averages of the partial cpu_i times are denoted by μ_i, $i = 1,2$. In terms of the total CPU, $\mu_t = \mu_1 + \mu_2$ and σ_t are used to represent the associate average and the standard deviation. The time measures are indexed by LA, RA or HA, according to the algorithm they are associated with.

The averages of the partial running times are reported in Table 2 for complete networks and in Tables 5 and 6 for random networks. Analogously, Table 3 for complete networks and Tables 7 and 8 for random networks show the averages and the standard deviations of the total CPU times. Some of the values are omitted when the codes were too slow.

Let N_r and N_h represent the total number of loopless $(1,n)$-paths returned by the ranking till a solution is found in RA and HA, respectively. The associate averages are denoted by μ_r and μ_h, respectively, whereas the correspondent standard deviations are σ_r and σ_h. Such results are presented in Table 4, for complete networks, and in Table 9, for random networks. These measures are sufficient to determine the averages and the standard deviations of the total number of computed loopless paths, $N_{p_r} = N_r + k$ for RA and $N_{p_h} = N_h + k$ for HA. Hence, the averages of N_{p_r} and N_{p_h} are given by $\mu_{p_r} = \mu_r + k$ and $\mu_{p_h} = \mu_h + k$, respectively, and, analogously, the associate standard deviations by $\sigma_{p_r} = \sigma_r$ and $\sigma_{p_h} = \sigma_h$. In the tables the values are rounded to the closest integer.

According to Tables 2, 5 and 6, computing the trees $\mathcal{T}^i$, $i \in I_k$, 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 ($k \leq 5$).

Generally speaking most of RA's time was invested on the second stage, namely when the number of ranked paths or the number of deviations costs was big enough to demand 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 $d = 15$. Indeed, 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 Table 4 shows. Moreover, since μ_r was always greater that μ_h, the results for $\mu_{2_{RA}}$ were always greater than $\mu_{2_{HA}}$, even for the cases where the second phase had a minor role in the performance in RA. This was the case of complete networks with $n = 5$ and $k < 100$, where in average up to 3 loopless paths were ranked, and of 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, like Tables 3, 7 and 8 show. 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 CPUs provide information about the variability on the results towards the correspondent averages. In this sense, LA and HA were the most stable codes, with standard deviations generally smaller than the ones correspondent to the associate averages. Instead, RA had the most irregular performance due to the high values of $\sigma_{2_{RA}}$, usually greater than $\mu_{2_{RA}}$. This is supported by the high variability of N_r on Tables 4 and 9. In contrast, N_h did not vary much and the averages μ_h are quite small, especially when $k \geq 50$ for denser networks, where only one iteration was needed to obtain the optimum solution.

The evolution of the average CPUs can be evaluated by varying a single parameter at a time. When n and d are fixed, Tables 2, 5 and 6 show that the average time in computing the trees $\mathcal{T}^i$, $i \in I_k$, 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, $\mu_{2_{LA}}$ and $\mu_{2_{HA}}$ showed the smoothest growths, which is in accordance with the balanced performances of LA and HA. Instead, $\mu_{2_{RA}}$ increased more irregularly with k, as a reflection of the unstable variation of μ_r in Tables 4 and 9.

As computing the trees $\mathcal{T}^i$, $i \in I_k$, is the common initial task for all algorithms, the behavior on their second phase mimics the evolution of their average total CPU times. The plots in Figures 8 and 9 show those growths in logarithmic scale for the three codes, when k varies on complete networks with n fixed and 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 $\mu_{t_{LA}}$ and $\mu_{t_{HA}}$ grew

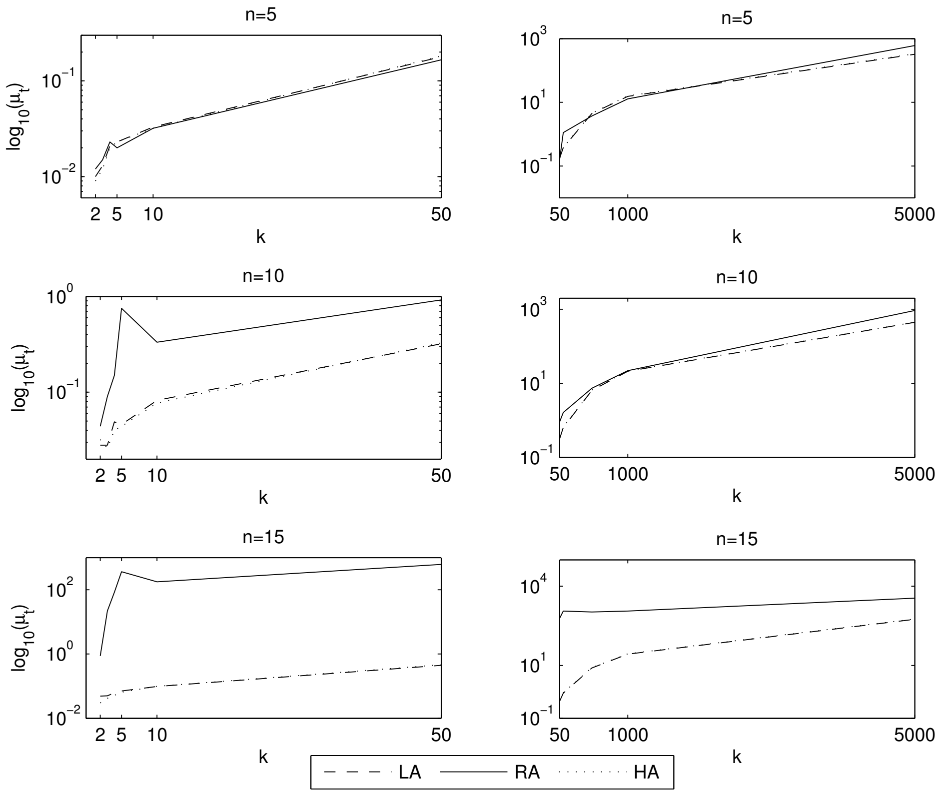


Figure 8: Performance of μ_t for complete networks with n fixed

similarly with the increase of k but slower than $\mu_{t_{RA}}$. The increase of the latter is steeper 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.

The obtained results may also be analyzed under the perspective of fixing the number of scenarios. Based on Figures 8 and 9, for different values of k the curves of $\mu_{t_{LA}}$ and $\mu_{t_{HA}}$ become more distant from the curve of $\mu_{t_{RA}}$ when n increases. This results from the growth of the number of paths in G with n, which may also affect 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 7 and 8, as well as by the plots in Figure 10 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 $n = 250$.

6 Conclusions

This work addressed the minmax regret robust shortest path problem on a finite number of scenarios. Some properties of this problem were derived and supported the development of three algorithms for finding an optimum 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, the pruning rules allow to

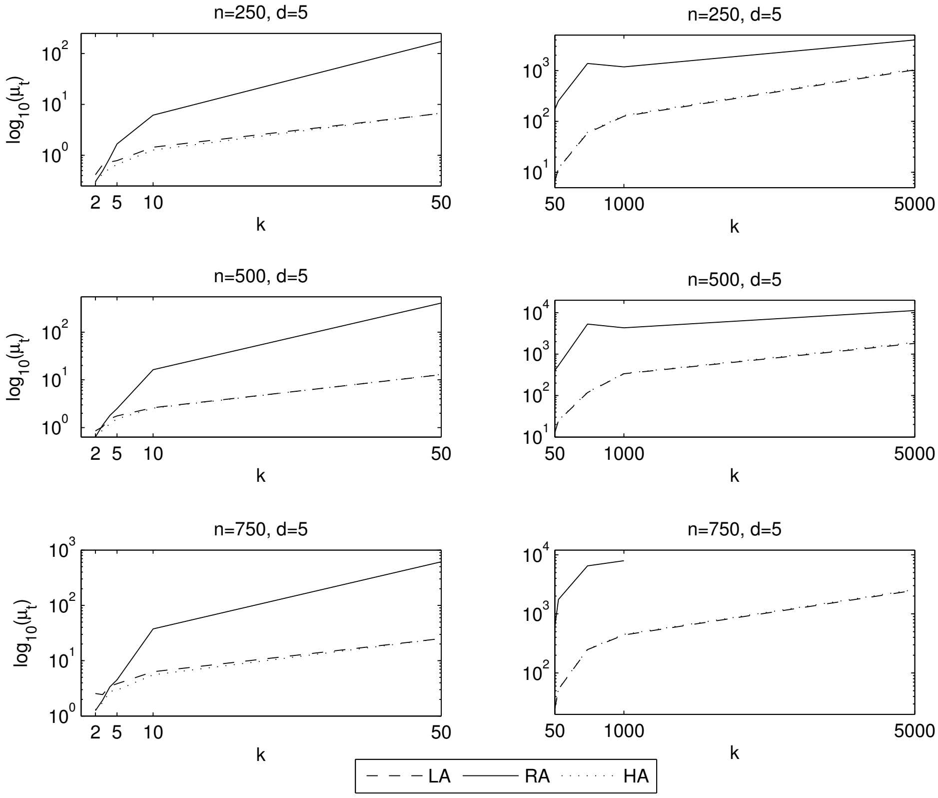


Figure 9: Performance of μ_t for random networks with $d = 5$ and n fixed

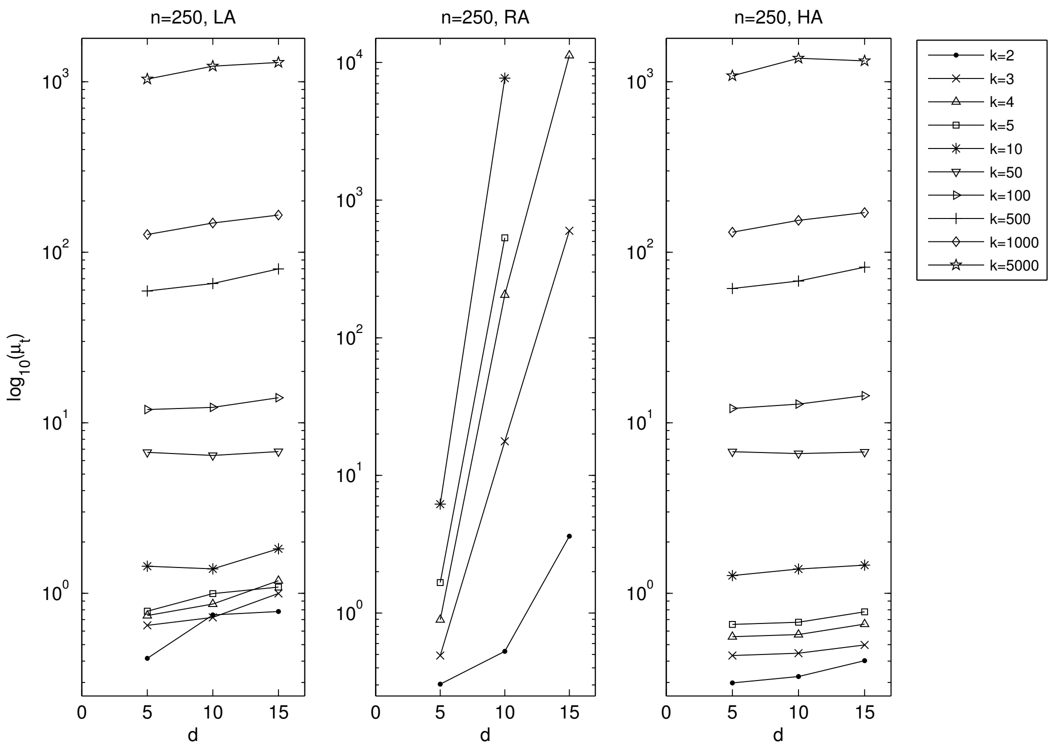


Figure 10: Performance of μ_t for random networks with $n = 250$ and k fixed

skip uninteresting paths; while, on the other, it assumes that a deviation ranking algorithm is used, and, thus, promotes the early generation of more candidate paths than with a standard implementation, seeking to produce good cost upper bounds. This further reinforces the elimination of bad solutions.

The time worst case computational complexities of the developed methods were deduced. The methods are pseudo-polynomial and their 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. The results of these experiments revealed that, in general, the ranking algorithm was the one with the poorest performance, due to the variable number of loopless paths that had to be ranked before obtaining the robust shortest path. 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. In the performed experiments the hybrid algorithm stood out when the number of scenarios did not exceed 100, solving the problem in less than one minute in average, whereas the labeling algorithm was the best for problems with 1000 or 5000 scenarios, running in less than one hour in average. Regarding previous literature it can be noted that the developed approaches showed to be quite effective for solving exactly the min-max regret robust shortest problem with up to 1000 scenarios.

Future research on this subject can be directed to explore further techniques for reducing the number of scenarios 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.

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.

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

$Q \leftarrow \emptyset$;
for $r = 1, \dots, k$ **do**
  Compute the tree $\mathcal{T}^r$ under t_r;
  $Q \leftarrow Q \cup \{p_1^r\}$;
  for $j = 1, \dots, n$ **do** $LB_j^r \leftarrow$ cost of the shortest (j, n)-path under t_r;
  ;
$UB \leftarrow \min\{RC(p_1^r) : r \in I_k\}$;
$sol \leftarrow p_1^r$ such that $RC(p_1^r) = UB$, $r \in I_k$;
$i \leftarrow \min\big\{r \in I_k : RD(sol, t_r) = UB\big\}$;
$Vmax \leftarrow v(sol, t_i)$;
$X \leftarrow \emptyset$; $p \leftarrow p_1^i$;
Store the network in the sorted forward star form with respect to the costs under t_i;
while *there exists a path p to be scanned such that $RD(p, t_i) \neq UB$* **do**
  $W \leftarrow \emptyset$;
  for $w \in V(p)$ *from the head node of p's deviation arc to the node that precedes n* **do**
    $p_{1w} \leftarrow (1, w)$-sub-path of p;
    if p_{1w} *is not loopless* **then break**;
    ;
    if $\max_{r \in I_k}\{v(p_{1w}, t_r) + LB_w^r - LB_1^r\} > UB$ **then break**;
    ;
    $x_1 \leftarrow$ head node of p's arc with tail node w;
    for $x \in \widehat{V}^{+i}(p_{1w}, x_1)$ **do**
      if $\max_{r \in I_k}\{v(p_{1w}, t_r) + c_{wx}^{r,k} + LB_x^r - LB_1^r\} \leq UB$ **then**
        $q_{w,x}^{p,i} \leftarrow p_{1w} \diamond \langle w, x\rangle \diamond p_x^{*i}$;
        if $v(q_{w,x}^{p,i}, t_i) > Vmax$ **then break**;
        ;
        $W \leftarrow W \cup \{q_{w,x}^{p,i}\}$;
        $RCaux \leftarrow \max\{v(p_{1w}, t_r) + c_{wx}^{r,k} + LB_x^r - LB_1^r : r \in I_k\}$;
        if $RCaux < UB$ **then**
          $UB \leftarrow RCaux$; $Vmax \leftarrow LB_1^i + UB$;
          Delete from W any path q such that $v(q, t_i) > Vmax$, or
          $\max_{r \in I_k}\{v(q_{1w}, t_r) + c_{wx}^{r,k} + LB_x^r - LB_1^r\} > UB$, with (w, x) the q's deviation arc;
        if $RD(q_{w,x}^{p,i}, t_i) = UB$ **then break**;
        ;
  if $RC(p) = UB$ **and** p is loopless **then** $sol \leftarrow p$;
  ;
  Delete from X any path q such that $v(q, t_i) > Vmax$, or
  $\max_{r \in I_k}\{v(q_{1w}, t_r) + c_{wx}^{r,k} + LB_x^r - LB_1^r\} > UB$, with (w, x) the q's deviation arc;
  $X \leftarrow X \cup W$;
  $p \leftarrow$ shortest path under t_i in X; $X \leftarrow X - \{p\}$;
return sol

j	Path p_j^1	$v(p_j^1, t_1)$	$v(p_j^1, t_2)$	Updates
1	$\langle 1, 2, 4, 6\rangle$	40	55	$UB \leftarrow 12$; $sol \leftarrow p_1^2$; $Vmax \leftarrow 52$
2	$\langle 1, 3, 5, 6\rangle$	45	52	$RCaux \leftarrow 12$;
3	$\langle 1, 3, 5, 4, 6\rangle$	45	51	$UB = RCaux \leftarrow 11 < 12$; $sol \leftarrow p_3^1$; $RD(p_3^1, t_1) \neq 11$; $Vmax \leftarrow 51$
4	$\langle 1, 3, 2, 4, 6\rangle$	50	50	$UB = RCaux \leftarrow 10 < 11$; $sol \leftarrow p_4^1$; $RD(p_4^1, t_1) = 10$; Stop

Table 1: Simulation of Algorithm 2

n	k	LA $\mu_{1_{LA}}^{(*)}$	LA $\mu_{2_{LA}}$	RA $\mu_{2_{RA}}$	HA $\mu_{2_{HA}}$
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

$^{(*)}: \mu_{1_{LA}} = \mu_{1_{RA}} = \mu_{1_{HA}}$

Table 2: Averages of the partial CPU times for complete networks

		LA		RA		HA	
n	k	$\mu_{t_{LA}}$	$\sigma_{t_{LA}}$	$\mu_{t_{RA}}$	$\sigma_{t_{RA}}$	$\mu_{t_{HA}}$	$\sigma_{t_{HA}}$
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 3: Averages and standard deviations of the total CPU times for complete networks

	$n=5$				$n=10$				$n=15$			
k	μ_r	σ_r	μ_h	σ_h	μ_r	σ_r	μ_h	σ_h	μ_r	σ_r	μ_h	σ_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 4: Averages and standard deviations of N_r and N_h for complete networks

			LA		RA	HA
n	d	k	$\mu_{1_{LA}}^{(*)}$	$\mu_{2_{LA}}$	$\mu_{2_{RA}}$	$\mu_{2_{HA}}$
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

$^{(*)}: \mu_{1_{LA}} = \mu_{1_{RA}} = \mu_{1_{HA}}$

Table 5: Averages of the partial CPU times for random networks with $n \in \{250, 500\}$

			LA		RA	HA
n	d	k	$\mu_{1_{LA}}^{(*)}$	$\mu_{2_{LA}}$	$\mu_{2_{RA}}$	$\mu_{2_{HA}}$
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

Table 6: Averages of the partial CPU times for random networks with $n = 750$

			LA		RA		HA	
n	d	k	$\mu_{t_{LA}}$	$\sigma_{t_{LA}}$	$\mu_{t_{RA}}$	$\sigma_{t_{RA}}$	$\mu_{t_{HA}}$	$\sigma_{t_{HA}}$
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
		1000	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
		1000	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
		1000	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
		1000	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
		1000	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
		1000	375.407	32.030	–	–	385.749	33.258
		5000	2071.345	105.449	–	–	2108.447	88.095

Table 7: Averages and standard deviations of the total CPU times for random networks with $n \in \{250, 500\}$

			LA		RA		HA	
n	d	k	$\mu_{t_{LA}}$	$\sigma_{t_{LA}}$	$\mu_{t_{RA}}$	$\sigma_{t_{RA}}$	$\mu_{t_{HA}}$	$\sigma_{t_{HA}}$
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 8: Averages and standard deviations of the total CPU times for random networks with $n = 750$

		$n = 250$				$n = 500$				$n = 750$			
d	k	μ_r	σ_r	μ_h	σ_h	μ_r	σ_r	μ_h	σ_h	μ_r	σ_r	μ_h	σ_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 9: Averages and standard deviations of N_r and N_h for random networks