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

Marta Pascoal and Marisa Resende

Dynamic preprocessing for the minmax regret robust shortest path problem with finite multi-scenarios

No.10 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

Dynamic preprocessing for the minmax regret robust shortest path problem with finite multi-scenarios

Marta M. B. Pascoal[*], Marisa Resende

September 22, 2014

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

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

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

Abstract

The minmax regret robust shortest path problem aims at finding a path that minimizes the maximum deviation from the shortest paths over all scenarios. It is assumed that different arc costs are associated with different arc scenarios. This paper introduces techniques to reduce the network, before a minmax regret robust shortest path algorithm is applied. The preprocessing methods enhance others explored in previous research. The introduced methods act dynamically and allow to update the conditions to be checked as new network arcs or network nodes that can be discarded are identified. Computational results on random networks are reported, which compare the dynamic preprocessing algorithms and their former static versions. Two robust shortest path algorithms are tested with and without these preprocessing rules.

Keywords: Robust shortest path, Discrete scenarios, Dynamic preprocessing.

1 Introduction

One approach for dealing with costs uncertainty is to consider several possible scenarios. In the case of the shortest path problem this is done either by associating a discrete set of costs with each arc, or by assuming each arc cost varies within an interval. In this paper, the former case is considered for the minimax regret robust shortest path problem, here simply called robust shortest path problem. This problem consists of finding a path between two nodes of a network, which minimizes the maximum regret cost of each path towards the shortest path, for all scenarios.

Yu and Yang [7] and, more recently, Pascoal and Resende [5], developed algorithms for the robust shortest path problem. Later, inspired by the works of Karasan, Pinar and Yaman [4] and then Catanzaro, Labbé and Salazar-Neumann [3], for the interval data case, Pascoal and Resende [6]

[*]Corresponding author

presented theoretical results and algorithms that allow to reduce the network before a robust shortest path algorithm is applied. These preprocessing techniques can identify arcs that are certainly part of an optimal solution, as well as identify a priori, and later delete, nodes that do not belong to any optimal solution.

The goal of this work is to enhance the preprocessing strategies developed in [6]. The improvements are a consequence of three aspects: a new result that combines two propositions presented in [6]; the development of dynamic preprocessing rules, in the sense that they are updated as the preprocessing algorithms run and paths are computed. The idea behind these improvements is to further reduce the network before a robust shortest path algorithm is applied, that is, to increase the number of detected arcs that belong, or the nodes that do not belong, to the optimal solutions. The last aspect concerns limiting the number of scenarios to consider in the tests, and thus to save computational time. Empirical experiments compare the new rules with the former.

The remainder of the paper contains five other sections. Notation and concepts related with the robust shortest path problem are introduced in the next one. Section 3 is dedicated to the development of the new preprocessing rules and of the algorithms to implement them. An example is provided in Section 4. Results of computational tests on random instances, comparing the new rules and their original static versions, when used together with the labeling and the hybrid approaches presented in [5], are reported and discussed in Section 5. Conclusions are drawn in Section 6.

2 Preliminary concepts

A finite multi-scenario model is represented as $G(V, A, S_k)$, where G is a directed graph with a set of nodes $V = \{1, \dots, n\}$, a set of m arcs $A \subseteq \{(i,j) : i,j \in V \text{ and } i \neq j\}$ and a finite set of scenarios $S_k := \{1, \dots, k\}$, $k > 1$. For each arc $(i,j) \in A$, c^s_{ij} represents its cost in scenario s, $s \in S_k$. It is assumed that the graph contains no parallel arcs. Let A' be a nonempty subset of A. Then, $G - A'$ denotes the subgraph of G with set of nodes V and set of arcs $A \backslash A'$. In particular, $G - \{(i,j)\}$ is represented by G^*_{ij}.

The set of arcs (nodes) in a path p is denoted by $A(p)$ $(V(p))$. 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 $p \diamond q$. The cost of a path p in scenario s, $s \in S_k$, is defined by

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

With no loss of generality, 1 and n denote the origin and the destination nodes of the graph G, respectively. The set of all $(1,n)$-paths in G is represented by $P(G)$.

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

The minmax regret robust shortest path problem aims at finding a path in $P(G)$ with the least maximum robust deviation, i.e., satisfying

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

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

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

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

$$RD^s(p) := c^s(p) - LB_1^s. \tag{4}$$

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

A node, or an arc, is called robust 1-persistent if it belongs to some robust shortest path. Otherwise, the node, or arc, is denominated robust 0-persistent. The origin and the destination nodes of the network are trivially robust 1-persistent nodes.

3 Preprocessing techniques

In [6], two sufficient conditions were established to identify robust 1-persistent arcs and robust 0-persistent nodes. The first condition restricts the arcs candidate to robust 1-persistent to those included in the shortest $(1, n)$-paths. The second condition is more global, as it allows to test all the nodes that do not belong to a given path in the network. These facts make the method based on the second condition more effective than the first for preprocessing the network. The computational experiments reported in [6] showed that the tests established for detecting robust 1-persistent arcs were effective very few times, even in networks with small density.

In this section new rules are developed to improve the previous preprocessing methods. One of them results from the combination of two rules introduced in [6], for detecting robust 1-persistent arcs and robust 0-persistent nodes. Another one consists of restricting the number of tested scenarios, whereas the remaining rules are dynamic approaches, in the sense that the tests are updated as new solutions are computed. These new rules allow to find a bigger number of robust 1-persistent arcs, and of robust 0-persistent nodes, than the previous.

For the sake of completeness, first, two results introduced in [6] are recalled to be used later. Proposition 1 concerns the identification of robust 1-persistent arcs, whereas Proposition 2 concerns the identification of robust 0-persistent nodes.

Proposition 1 ([6]). *Let $q \in P(G)$ be a path such that $A(q) \cap \{A(p^{1,s}) : s \in S_k\} \neq \emptyset$, and $(i, j) \in A(q) \cap \{A(p^{1,s}) : s \in S_k\}$ be an arc such that node n is reachable from node 1 in G^*_{ij}. Let $S(i, j) = \{s \in S_k : (i, j) \in p^{1,s}\}$ be the set of scenarios for which the shortest $(1, n)$-paths contain arc (i, j) and $p^{1,s}_{*ij}$ denote the shortest $(1, n)$-path in $P(G^*_{ij})$ in scenario s, $s \in S_k$. If*

$$\exists \hat{s} \in S(i, j) : \; RD^{\hat{s}}(p^{1,\hat{s}}_{*ij}) > RC(q), \tag{5}$$

then arc (i, j) is robust 1-persistent.

Proposition 2 ([6]). *Consider a path $q \in P(G)$, and a node $i \notin V(q)$. If*

$$\exists \hat{s} \in S_k : \; RD^{\hat{s}}(p^{1,\hat{s}}_{1i} \diamond p^{1,\hat{s}}_{in}) > RC(q), \tag{6}$$

then node i is robust 0-persistent.

The combination of Propositions 1 and 2 can further enhance the former preprocessing rules. Even though the search for robust 1-persistent arcs is confined to the set of arcs of the shortest $(1, n)$-paths of the network, identifying one of them makes easier the detection of robust 0-persistent nodes. In fact, under such circumstances, the robust deviations calculation can be avoided. This result is stated in the following.

Corollary 1. *Let arc (i, j) be a robust 1-persistent, and let $q \in P(G)$ be a path, with $(i, j) \in A(q) \cap \{A(p^{1,s}) : s \in S_k\}$. Then, any node $j' \notin V(q)$ such that $(i, j) \notin p_{1j'}^{1,\hat{s}}$ and $(i, j) \notin p_{j'n}^{1,\hat{s}}$, for some $\hat{s} \in S(i, j)$, is robust 0-persistent.*

Proof. If arc (i, j) is robust 1-persistent and $(i, j) \in A(q) \cap \{A(p^{1,s}) : s \in S_k\}$ for some $q \in P(G)$, then, according to Proposition 1, (5) is satisfied for some $\hat{s} \in S(i, j)$. Since $p_{*ij}^{1,\hat{s}}$ represents the shortest $(1, n)$-path for scenario $\hat{s}$ in G that does not contain arc (i, j), then, any other path $q' \in P(G)$, such that $(i, j) \notin A(q')$, satisfies $c^{\hat{s}}(q') \geq c^{\hat{s}}(p_{*ij}^{1,\hat{s}})$, and, from (5),

$$RD^{\hat{s}}(q') \geq RD^{\hat{s}}(p_{*ij}^{1,\hat{s}}) > RC(q). \tag{7}$$

Hence, any node $j' \notin V(q)$ such that $(i, j) \notin p_{1j'}^{1,\hat{s}}$ and $(i, j) \notin p_{j'n}^{1,\hat{s}}$ makes that $p_{1j'}^{1,\hat{s}} \diamond p_{j'n}^{1,\hat{s}}$ does not contain arc (i, j), and, therefore, q' can be set to that $(1, n)$-path. Thus, (6) follows from (7), and according to Proposition 2 this means that j' is a robust 0-persistent node. □

Some results are now presented to support an algorithm for identifying robust 1-persistent arcs and another one for identifying robust 0-persistent nodes. As mentioned earlier, the idea behind these versions is to make the search dynamic and detect robust 1-persistent arcs and robust 0-persistent nodes, according to the least robustness cost of the $(1, n)$-paths obtained along the process.

Let $RCmin$ be a variable which stores the least robustness cost of a computed $(1, n)$-path at any iteration of the algorithms. That variable is initialized with

$$RCmin = \min\{RC(p^{1,s}); s \in S_k\},$$

for detecting robust 1-persistent arcs, and, considering only the shortest $(1, n)$-path in scenario 1, $p^{1,1,}$, with

$$RCmin = RC(p^{1,1}),$$

for identifying robust 0-persistent nodes. Let Arc and Nod denote the sets of arcs and of nodes to be scanned, respectively. The conditions provided by Propositions 1 and 2 can be rewritten, using variable $RCmin$. For any arc $(i, j) \in Arc$, if node n is reachable from node 1 in G_{ij}^* and

$$\exists \hat{s} \in S(i, j) \; : \; RD^{\hat{s}}(p_{*ij}^{1,\hat{s}}) > RCmin, \tag{8}$$

holds, then the arc (i, j) is robust 1-persistent. Similarly, for any node $i \in Nod$, if

$$\exists \hat{s} \in S_k \; : \; RD^{\hat{s}}(p_{1i}^{1,\hat{s}} \diamond p_{in}^{1,\hat{s}}) > RCmin, \tag{9}$$

is satisfied, then the node i is robust 0-persistent. These conditions demand the tree of the shortest (j, n)-paths for each scenario s, denoted by $\mathcal{T}^s$, $j \in V$, $s \in S_k$, and their costs LB_j^s, to be known. Any shortest path tree algorithm can be used with such purpose [1].

Let A_1 (V_0) be used to collect the robust 1-persistent arcs (0-persistent nodes). Let also Q be the set of the shortest $(1,n)$-paths with the minimum robustness cost, i.e.

$$Q = \{p^{1,s} : s \in S_k \text{ and } RC(p^{1,s}) = RCmin\}.$$

According to Propositions 1 and 2, and to the initialization of $RCmin$, Arc and Nod are initialized by

$$Arc = \Big\{(i,j) \in A(q) : q \in Q\Big\},$$

and

$$Nod = V \backslash V(p^{1,1}).$$

The value of variable $RCmin$ may change along the algorithms. The $(1,n)$-paths computed by the algorithms are stored in a list X_P, without repetitions. The set of arcs/nodes to scan may also change, every time a new $(1,n)$-path q such that $q \notin X_P$ has a robustness cost not greater than $RCmin$. If $RC(q) < RCmin$, $RCmin$ is updated with $RC(q)$. In what follows, it is shown how to update Arc and Nod, depending on the obtained path q satisfying $RC(q) \leq RCmin$.

When identifying robust 1-persistent arcs, Proposition 1 allows to establish that if $RC(q) = RCmin$, the arcs of $\{A(q) \cap (p^{1,s}) : s \in S_k\}$ that were not identified as robust 1-persistent, i.e., those in $\{A(q) \cap A(p^{1,s}) : s \in S_k\} \backslash A_1$, should be analyzed. In case $RC(q) < RCmin$, the search focuses only on the previous set, since path q is a new candidate for the optimal solution. For a selected $(i,j) \in Arc$, path q takes the particular form $p^{1,s}_{*ij}$, for some $s \in S(i,j)$. Under these conditions, one can write

$$Arc = \begin{cases} Arc \cup \big(\{A(p^{1,s}_{*ij}) \cap A(p^{1,s'}) : s' \in S_k\} \backslash A_1\big) & \text{if } \; RC(p^{1,s}_{*ij}) = RCmin \\ \{A(p^{1,s}_{*ij}) \cap A(p^{1,s'}) : s' \in S_k\} \backslash A_1 & \text{if } \; RC(p^{1,s}_{*ij}) < RCmin \end{cases} \tag{10}$$

When searching for robust 0-persistent nodes, Proposition 2 establishes that the analysis of the nodes of path q, $V(q)$, can be skipped. Thus, if $RC(q) = RCmin$, the nodes of $V(q)$ can be removed from Nod, and, if $RC(q) < RCmin$, the search focuses all the nodes outside $V(q)$ that were not already identified as robust 0-persistent. For a selected node $i \in Nod$, path q has the particular form $p^{1,s}_{1i} \diamond p^{1,s}_{in}$, $s \in S_k$. Then, one can write

$$Nod = \begin{cases} Nod \backslash V(p^{1,s}_{1i} \diamond p^{1,s}_{in}) & \text{if } \; RC(p^{1,s}_{1i} \diamond p^{1,s}_{in}) = RCmin \\ V \backslash (V(p^{1,s}_{1i} \diamond p^{1,s}_{in}) \cup V_0) & \text{if } \; RC(p^{1,s}_{1i} \diamond p^{1,s}_{in}) < RCmin \end{cases} \tag{11}$$

Arcs/nodes may be scanned more than once, because the analyzed $(1,n)$-paths may have arcs/nodes in common. This makes that some tests may be repeated after $RCmin$ is updated. Besides, in order to avoid repeating the path robust deviations, it is useful to store them, as

$$RD^s_{*ij} = RD^s(p^{1,s}_{*ij}) \, , \; s \in S_k \, , \; (i,j) \in A, \text{ such that } p^{1,s}_{*ij} \text{ exists in } G^*_{ij}, \tag{12}$$

and

$$RD^s_i = RD^s(p^{1,s}_{1i} \diamond p^{1,s}_{in}) \, , \; s \in S_k \, , \; i \in V \backslash \{1,n\}. \tag{13}$$

A list X_A/X_N is used to store the arcs/nodes that have already been analyzed along the process.

The dynamic procedure for identifying robust 1-persistent arcs is outlined in Algorithm 1.

Algorithm 1: Dynamic version for finding robust 1-persistent arcs

```
for $s = 1, \dots, k$ do
    $p^{1,s} \leftarrow$ shortest path for scenario $s$;
    $LB_1^s \leftarrow c^s(p^{1,s})$;
$RCmin \leftarrow \min\{RC(p^{1,s}) : s \in S_k\}$;
$Q \leftarrow \{p_1^s : s \in S_k$ and $RC(p^{1,s}) = RCmin\}$;
$X_P \leftarrow \{p^{1,s} : s \in S_k\}$;
$Arc \leftarrow \{(i,j) \in A(q) : q \in Q\}$;
$X_A \leftarrow \emptyset$ ; $A_1 \leftarrow \emptyset$;
while $Arc \neq \emptyset$ do
    Choose an arc $(i,j) \in Arc$;
    $Arc \leftarrow Arc - \{(i,j)\}$;
    if $(i,j) \notin X_A$ and node $n$ is reachable from node 1 in $G_{ij}^*$ then
        $X_A \leftarrow X_A \cup \{(i,j)\}$;
        $S(i,j) \leftarrow \{s \in S_k : (i,j) \in p^{1,s}\}$;
        for $s \in S(i,j)$ do
            $p_{*ij}^{1,s} \leftarrow$ shortest path for scenario $s$ in $G_{ij}^*$;
            $RD_{*ij}^s \leftarrow RD^s(p_{*ij}^{1,s})$;
            if $RD_{*ij}^s > RCmin$ then
                $A_1 \leftarrow A_1 \cup \{(i,j)\}$;
                break;
            if $p_{*ij}^{1,s} \notin X_P$ then
                $X_P \leftarrow X_P \cup \{p_{*ij}^{1,s}\}$;
                $RC(p_{*ij}^{1,s}) \leftarrow \max\left\{RD_{*ij}^s, \max\left\{RD^r(p_{*ij}^{1,s}) : r \in S_k \backslash \{s\}\right\}\right\}$;
                if $RC(p_{*ij}^{1,s}) = RCmin$ then $Arc \leftarrow Arc \cup \left(\{A(p_{*ij}^{1,s}) \cap A(p^{1,s'}) : s' \in S_k\} \backslash A_1\right)$;
                if $RC(p_{*ij}^{1,s}) < RCmin$ then
                    $RCmin \leftarrow RC(p_{*ij}^{1,s})$;
                    $Arc \leftarrow \{A(p_{*ij}^{1,s}) \cap A(p^{1,s'}) : s' \in S_k\} \backslash A_1$;
    else
        for $s \in S(i,j)$ do
            if $RD_{*ij}^s > RCmin$ then
                $A_1 \leftarrow A_1 \cup \{(i,j)\}$;
                break;
return $A_1$
```

Algorithm 1 performs three additional tasks, when compared to the static version of the method for identifying robust 1-persistent arcs presented in [6]. Namely, the calculation of the robustness costs of the $(1,n)$-paths $p^{1,s}_{*ij}$, $(i,j) \in Arc$, $s \in S(i,j)$, the updates of set Arc, and the repetition of the tests (8) consequent from the update of $RCmin$. For the first task, assuming the trees $\mathcal{T}^s$ and the costs LB^s_j, $j \in V$, $s \in S_k$ were previously computed, the robustness cost of $p^{1,s}_{*ij}$, $(i,j) \in Arc$, $s \in S(i,j)$, is obtained in $\mathcal{O}(kn)$ time. In what concerns the second procedure, the update of Arc by (10) is made through intersections, unions and differences of subsets in $\{A(p^{1,s}) : s \in S_k\}$, which has $k(n-1)$ arcs at most. An efficient way to make such operations is with indexation using hash sets [2], which is of $\mathcal{O}(N)$, where N is the total number of elements. Hence, an $\mathcal{O}(kn)$ complexity is obtained. For the third task, when test (8) is repeated for a given arc $(i,j) \in Arc$, $\mathcal{O}(k)$ operations are required at most, one per each scenario $s \in S(i,j)$, because RD^s_{*ij} was already determined. In a worst case, the tasks above are performed $k^2(n-1)$ times at most, one per each scenario in S_k and each arc selected in Arc, with up to $k(n-1)$ elements. Consequently, the number of operations performed by the static procedure, $\mathcal{O}(k^2mn)$ for acyclic networks and $\mathcal{O}(k^2mn + k^2n^2 \log n)$ for general networks [6], increases by $\mathcal{O}(k^3n^2)$. Therefore, Algorithm 1 performs in $\mathcal{O}(k^2mn + k^3n^2)$ time for acyclic networks and in $\mathcal{O}(k^2mn + k^2n^2 \log n + k^3n^2)$ time for general networks.

The number of scenarios used to test condition (6) may make the robust 0-persistent nodes test computationally demanding. In [6] this test uses k scenarios. The same holds for condition (9), so in order to make this task lighter, in the following only a small number of scenarios to test M, $M \leq k$, will be considered. Moreover, for each node $i \in Nod$, when the first scenario $s_i \in S_k$ for which (9) holds is known, then i is a robust 0-persistent node and its analysis can halt. Hence, the tests for scenarios $s_i + 1, \ldots, M$, if $s_i \neq M$, can be skipped. Generally, if $\max\{s_i : i \in Nod\} \neq M$, the computation of the trees $\widetilde{\mathcal{T}}^s$ can be skipped for $s \in \{\max\{s_i : i \in Nod\}, \ldots, M\}$. The pseudo-code is given in Algorithm 2.

The static version of Algorithm 2 has time of $\mathcal{O}(kmn + kn^2 + k^2n)$ for acyclic networks and of $\mathcal{O}(kmn + kn^2 \log n + k^2n)$ for general networks [6]. In terms of the worst case computational time complexity, the first phase of Algorithm 2 is similar to the first phase of the former version. The second phase concerns the search for robust 0-persistent nodes, which compared to the static version has the additional work of calculating the robustness costs of the $(1,n)$-paths $p^{1,s}_{1i} \diamond p^{1,s}_{in}$, $i \in Nod$, $s \in S_k$, updating set Nod, and repeating the tests (9) due to the updates of $RCmin$. For the first task, assuming the trees $\mathcal{T}^s$ and $\widetilde{\mathcal{T}}^s$ and the associate costs were previously computed, the robustness cost of $p^{1,s}_{1i} \diamond p^{1,s}_{in}$, $i \in Nod$, $s \in S_k$, is obtained in $\mathcal{O}(k)$ time. The second task concerns the update of Nod by (11), and involves differences and unions of sets with n nodes at most. Hence, these operations require an $\mathcal{O}(n)$ complexity, when using indexation through hash sets [2]. The third procedure, demands k operations at most, one per each scenario $s \in S_k$, since RD^s_i was already determined, and, therefore, it has $\mathcal{O}(k)$ complexity. In a worst case, the three tasks are performed $k(n-2)$ times at most, one per each scenario in S_k and each selected node in Nod, with up to $n-2$ nodes. Thus, an additional work of $\mathcal{O}(kn^2)$ is added to the second phase of the former version. Nevertheless, this does not affect the total complexities of the static version.

Finally, it should be noted that all the robust 1-persistent arcs and the robust 0-persistent nodes identified with the static approaches are still identified with the dynamic approaches.

Algorithm 2: Dynamic version for finding robust 0-persistent nodes

```
for $s = 1, \ldots, k$ do
    Compute the tree $\mathcal{T}^s$;
    for $j = 1, \ldots, n$ do  $LB^s_j \leftarrow c^s(p^{1,s}_{jn})$;
$RCmin \leftarrow RC(p^{1,1})$;
$X_P \leftarrow \{p^{1,1}\}$ ; $X_N \leftarrow \emptyset$;
$Nod \leftarrow V \backslash V(p^{1,1})$ ; $V_0 \leftarrow \emptyset$;
while $Nod \neq \emptyset$ do
    Choose a node $i \in Nod$;
    $Nod \leftarrow Nod - \{i\}$;
    if $i \notin X_N$ then
        $X_N \leftarrow X_N \cup \{i\}$;
        for $s = 1, \ldots, M$ do
            if tree $\widetilde{\mathcal{T}}^s$ was not yet determined then  Compute the tree $\widetilde{\mathcal{T}}^s$;
            $RD^s_i \leftarrow c^s(p^{1,s}_{1i}) + LB^s_i - LB^s_1$;
            if $RD^s_i > RCmin$ then
                $V_0 \leftarrow V_0 \cup \{i\}$;
                break;
            if $p^{1,s}_{1i} \diamond p^{1,s}_{in} \notin X_P$ then
                $X_P \leftarrow X_P \cup \{p^{1,s}_{1i} \diamond p^{1,s}_{in}\}$;
                $RC(p^{1,s}_{1i} \diamond p^{1,s}_{in}) \leftarrow \max\left\{RD^s_i, \max\left\{RD^r(p^{1,s}_{1i} \diamond p^{1,s}_{in}) : r \in S_k \backslash \{s\}\right\}\right\}$;
                if $RC(p^{1,s}_{1i} \diamond p^{1,s}_{in}) = RCmin$ then  $Nod \leftarrow Nod \backslash V(p^{1,s}_{1i} \diamond p^{1,s}_{in})$;
                if $RC(p^{1,s}_{1i} \diamond p^{1,s}_{in}) < RCmin$ then
                    $RCmin \leftarrow RC(p^{1,s}_{1i} \diamond p^{1,s}_{in})$;
                    $Nod \leftarrow V \backslash (V(p^{1,s}_{1i} \diamond p^{1,s}_{in}) \cup V_0)$;
    else
        for $s = 1, \ldots, M$ do
            if $RD^s_i > RCmin$ then
                $V_0 \leftarrow V_0 \cup \{i\}$;
                break;
return $V_0$
```

4 Example

In the following, the dynamic algorithms for finding robust 1-persistent arcs and robust 0-persistent nodes introduced in Section 3 are exemplified. In order to better understand the differences introduced in the previous algorithms with respect to the static preprocessing methods presented in [6], the application of the two approaches is described.

Let $G(V, A, S_2)$ be the network depicted in Figure 1. Figure 2 shows the shortest path trees from every node to node 7 in G, in scenario 1 – Figure 2.(a) – and in scenario 2 – Figure 2.(b).

Identifying robust 1-persistent arcs According to Figure 2, Q is set to $\{p^{1,1}, p^{1,2}\}$, with $p^{1,1} = \langle 1, 2, 7\rangle$, $LB^1_1 = 2$, and $p^{1,2} = \langle 1, 4, 6, 7\rangle$, $LB^2_1 = 7$. Given that $c^2(p^{1,1}) = 12$ and $c^1(p^{1,2}) = 8$, one has $RC(p^{1,1}) = 5$ and $RC(p^{1,2}) = 6$. Hence, $p^{1,1}$ is the path with the least robustness cost in Q, and, therefore, Arc and $RCmin$ are initialized with $\{(1, 2), (2, 7)\}$ and 5, respectively, for both

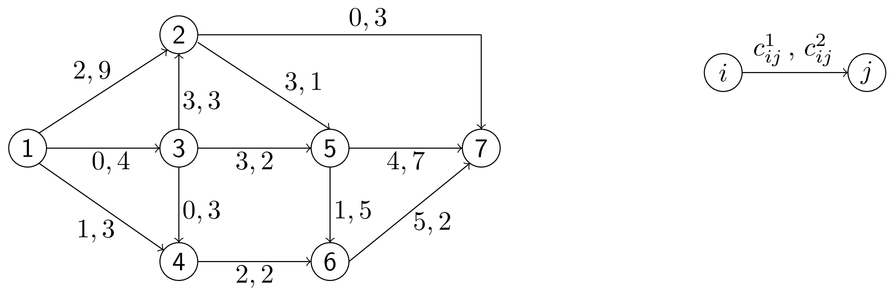

Figure 1: Network $G(V, A, S_2)$

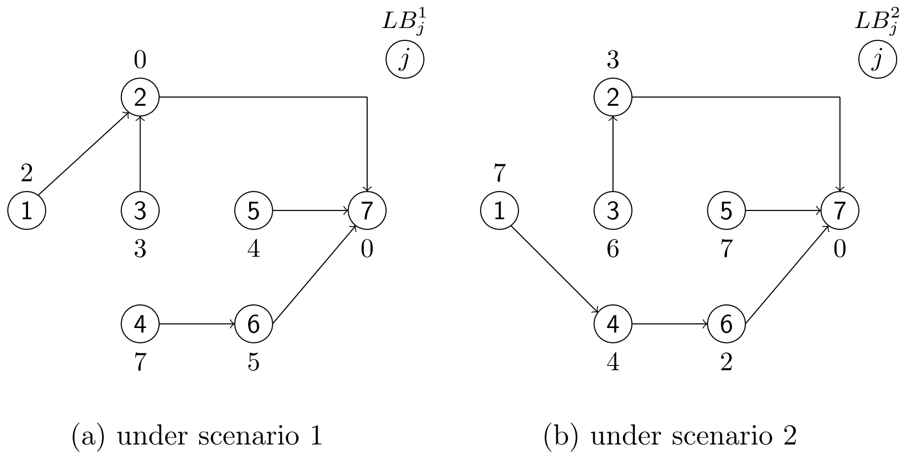

Figure 2: Shortest path trees rooted at node 7 in $G(V, A, S_2)$

approaches. The latter value does not change in the static method.

Static approach First, the arc $(1, 2)$ is considered and $S(1, 2) = \{1\}$. Node 7 is reachable from node 1 in G^*_{12} and $p^{1,1}_{*12} = \langle 1, 3, 2, 7\rangle$, with

$$RD^1(p^{1,1}_{*12}) = c^1(p^{1,1}_{*12}) - LB^1_1 = 1.$$

Therefore, as $RCmin = 5$, condition (8) is not satisfied, and nothing can be concluded concerning arc $(1, 2)$. Afterwards, arc $(2, 7)$ is selected and $S(2, 7) = \{1\}$. Now, node 7 is reachable from node 1 in G^*_{27} and $p^{1,1}_{*27} = \langle 1, 3, 5, 7\rangle$, with

$$RD^1(p^{1,1}_{*27}) = c^1(p^{1,1}_{*27}) - LB^1_1 = 5.$$

Once again, condition (8) is not satisfied, thus no robust 1-persistent arcs are detected, i.e. $A_1 = \emptyset$.

Dynamic approach Like before, Algorithm 1 first scans arc $(1, 2)$, which does not satisfy (8) for $RCmin = 5$. Additionally, the robustness cost of $p^{1,1}_{*12} = \langle 1, 3, 2, 7\rangle$ is determined by

$$RC(p^{1,1}_{*12}) = \max_{s\in S_2}\{1, RD^2(\langle 1, 3, 2, 7\rangle)\} = \max_{s\in S_2}\{1, c^2(\langle 1, 3, 2, 7\rangle) - LB^2_1\} = 3.$$

which improves $RCmin$ to 3. Therefore, because $A_1 = \emptyset$, according to (10) Arc is updated to

$$Arc = \{A(\langle 1, 3, 2, 7\rangle) \cap A(p^{1,s}) : s \in S_2\} = \{(2, 7)\}.$$

Then, arc $(2,7)$ is selected and $p^{1,1}_{*27} = \langle 1,3,5,7 \rangle$, with $RD^1(p^{1,1}_{*27}) = 5$. For the updated $RCmin = 3$, the condition (8) holds, which means that arc $(2,7)$ is robust 1-persistent, i.e. $A_1 = \{(2,7)\}$.

Identifying robust 0-persistent nodes Figure 3 shows the shortest path trees from node 1 to any node in $G(V, A, S_2)$, in scenario 1 – Figure 3.(a) – and in scenario 2 – Figure 3.(b).

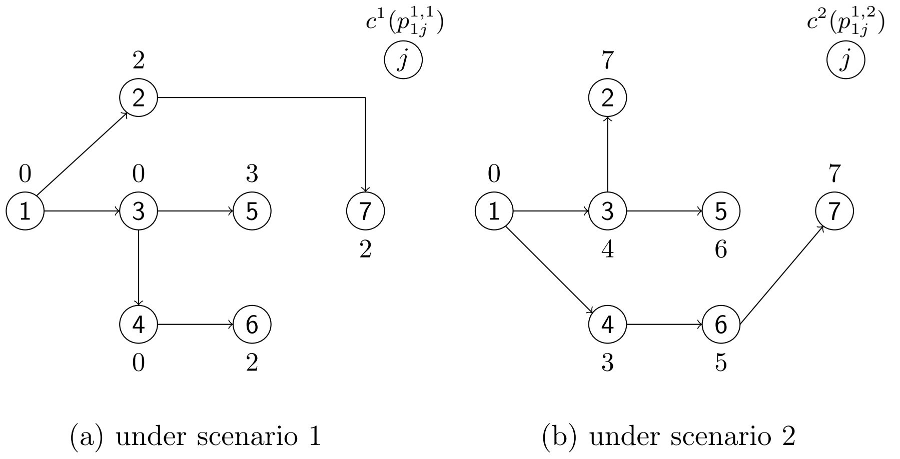

Figure 3: Shortest path trees rooted at node 1 in $G(V, A, S_2)$

In what follows the number of scenarios tested in (8) is limited to $M \in \{1, 2\}$. As mentioned in the previous section, this constraint was not used in [6], it will be applied to both approaches for the sake of comparing them.

Static approach Because $p^{1,1} = \langle 1,2,7 \rangle$ is the shortest $(1,n)$-path with the least robustness cost, 5, Nod is initialized with $\{3,4,5,6\}$ and $RCmin = 5$.

- $M = 1$

  Starting with node 3, the inequality (9) is not satisfied in scenario 1, given that

$$RD^1_3 = c^1(p^{1,1}_{13}) + LB^1_3 - LB^1_1 = 1 \le RCmin.$$

  The same thing happens for nodes 4,5 and 6, because

$$RD^1_i = c^1(p^{1,1}_{1i}) + LB^1_i - LB^1_1 = 5 \le RCmin\,,\ i = 4,5,6.$$

  Therefore, no robust 0-persistent nodes are detected when considering only scenario 1, $V_0 = \emptyset$.

- $M = 2$

  For scenario 2 the nodes 3, 4 and 6 still do not satisfy (9), given that

$$RD^2_3 = c^2(p^{1,2}_{13}) + LB^2_3 - LB^2_1 = 3 \le RCmin,$$

  and

$$RD^2_i = c^2(p^{1,2}_{1i}) + LB^2_i - LB^2_1 = 0 \le RCmin\,,\ i = 4,6.$$

Nevertheless, (9) holds for node 5 and scenario 2,

$$RD_5^2 = c^2(p_{15}^{1,2}) + LB_5^2 - LB_1^2 = 6 > RCmin,$$

therefore, node 5 is the only one identified as robust 0-persistent, $V_0 = \{5\}$.

Dynamic approach In this case, $RCmin$ is initialized with $RC(p^{1,1}) = 5$ and Nod with $V \backslash V(p^{1,1}) = \{3, 4, 5, 6\}$.

- $M = 1$

  Starting by scanning node 3, condition (9) is not satisfied for scenario 1. Then, the robustness cost of the path $p_{13}^{1,1} \diamond p_{37}^{1,1} = \langle 1, 3, 2, 7\rangle$ is determined, $RC(\langle 1, 3, 2, 7\rangle) = 3$, and this value improves $RCmin$. Additionally, by (11) Nod is updated to $V \backslash V(\langle 1, 3, 2, 7\rangle) = \{4, 5, 6\}$, since at this point $V_0 = \emptyset$.

  For the updated $RCmin = 3$, when choosing nodes 4, 5 and 6 to scan, inequality (9) is always satisfied for scenario 1, given that

  $$RD_i^1 = 5 > RCmin\,,\ i = 4, 5, 6.$$

  Consequently, all the nodes in Nod are identified as robust 0-persistent, i.e., $V_0 = \{4, 5, 6\}$.

- $M = 2$

  Condition (9) holds for node 3 and scenarios 1 and 2, with the initial $RCmin = 5$. Then, the path associated with node 3 for scenarios 1 and 2, $p_{13}^{1,s} \diamond p_{37}^{1,s}$, $s \in S_2$, is given by $\langle 1, 3, 2, 7\rangle$, which has a robustness cost of 3. The remaining steps are those presented for $M = 1$, thus $V_0 = \{4, 5, 6\}$.

Finally, Corollary 1 is applied to detect robust 0-persistent nodes. The arc $(2, 7)$ was identified above as robust 1-persistent, when associated to path $q = \langle 1, 3, 2, 7\rangle$, with $S(2, 7) = \{1\}$. According to Figures 2 and 3, one has $p_{14}^{1,1} = \langle 1, 3, 4\rangle$, $p_{47}^{1,1} = \langle 4, 6, 7\rangle$, $p_{15}^{1,1} = \langle 1, 3, 5\rangle$, $p_{57}^{1,1} = \langle 5, 7\rangle$, $p_{16}^{1,1} = \langle 1, 3, 4, 6\rangle$ and $p_{67}^{1,1} = \langle 6, 7\rangle$, and none of these sub-paths contains arc $(2, 7)$, i.e. $(2, 7) \notin p_{1i}^{1,1}$ and $(2, 7) \notin p_{in}^{1,1}$, for every $i \in V \backslash V(q) = \{4, 5, 6\}$. Therefore, $V_0 = \{4, 5, 6\}$.

For this example Algorithm 2 is more effective than its static version, given that it detects more robust 0-persistent nodes than the former version and the same set of nodes as Corollary 1, even when $M = 1$. The applications of the previous dynamic rules and of Corollary 1 are independent. Besides, the above results show that the dynamic rules can be a good alternative method for preprocessing when Corollary 1 cannot be applied, that is, when no robust 1-persistent arcs have been detected.

Computing a robust shortest path after preprocessing The reduced network obtained from preprocessing is depicted in Figure 4. Arc $(2, 7)$ must be contained in the optimum solution since it is robust 1-persistent. Thus, the reduced network results from removing from G all the remaining arcs that start in node 2, $(2, 5)$, or that end in node 7, $(5, 7)$ and $(6, 7)$. At this moment nothing can be said about the other arcs in G, represented with a dashed line in Figure 4. The robust

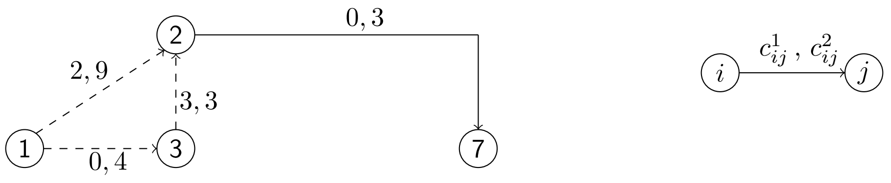

Figure 4: Reduced network after preprocessing

0-persistent nodes, 4, 5 and 6, are also removed from G, as well as all the arcs that start or end in these nodes.

There are only two $(1, 7)$-paths containing arc $(2, 7)$ in the reduced network, $p^{1,1} = \langle 1, 2, 7\rangle$, with $RC(p^{1,1}) = 5$, and $q = \langle 1, 3, 2, 7\rangle$, with $RC(q) = 3$. Therefore, q is the robust shortest path in G.

5 Computational experiments

This section is dedicated to the computational comparison of the static (presented in [6]) and the dynamic (in Algorithm 2) methods for preprocessing robust 0-persistent nodes, and to their impact on solving the robust shortest path problem when combined with the labeling and the hybrid algorithms introduced in [5]. The reason for not considering the empirical results for preprocessing robust 1-persistent arcs is that the methods developed with that purpose only showed to be effective for networks with a very small density $d = m/n$, $d \in \{1, 2\}$. In fact, in these cases the majority of the arcs of the network belongs to the shortest $(1, n)$-paths $p^{1,s}$, $s \in S_k$, which improves the chances of finding robust 1-persistent arcs.

Algorithm 2 and its static version were implemented in Matlab 7.12 and ran on a computer equipped with an Intel Pentium Dual CPU T2310 1.46GHz processor and 2GB of RAM. The codes use Dijkstra's algorithm [1] to solve the single destination shortest path problem for a given scenario. As mentioned earlier, the preprocessing techniques were combined with the labeling algorithm (LA) and the hybrid algorithm (HA) in [5]. The robust shortest path problem was solved with and without preprocessing.

The benchmarks used in the experiments correspond to randomly generated directed graphs with $n \in \{500, 1000, 2000, 5000\}$ nodes, density $d \in \{5, 10, 20\}$, and $k \in \{2, 3\}$ scenarios. For each scenario, each arc cost is assigned with a random integer number in $U(0, 100)$.

For each network dimension, 10 instances were generated. For each instance, the two preprocessing algorithms were applied, and (9) was tested for the scenarios $1, \dots, M$, with $M \in \{1 \dots k\}$. The robust shortest path problems were solved by LA and by HA, after preprocessing. Alternatively, LA and HA solved the same instances from scratch, with no preprocessing.

5.1 Results

In order to analyze the performance of the static and the dynamic algorithms, the average total running times (in seconds) are calculated for each network dimension. Let P_0, NP and AP_0 represent the average CPU times to preprocess robust 0-persistent nodes, to solve the robust shortest path problem with no preprocessing, and to do the same after preprocessing, respectively. Let also TP_0 denote the average overall CPU time for finding a robust shortest path combined with preprocessing, i.e., $TP_0 = P_0 + AP_0$. Additionally, let N_0 represent the number of robust 0-persistent nodes. The application of the static and the dynamic methods is distinguished by the indices s and d, respectively.

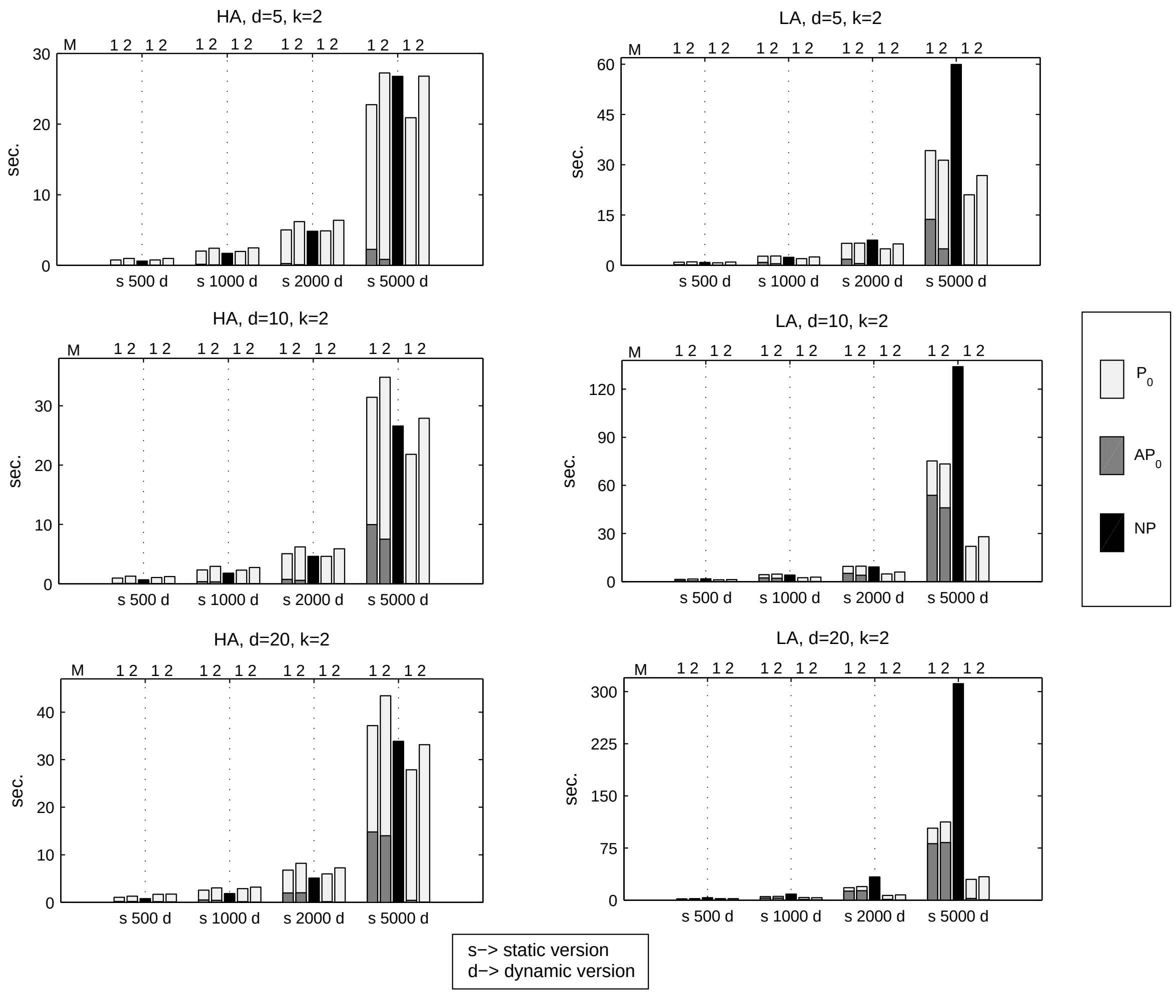

Figure 5: Average CPU times for preprocessing robust 0-persistent nodes and for algorithms HA and LA, with and without preprocessing, when $k = 2$

The average CPU times and the number of robust 0-persistent nodes are reported in Tables 1, 2 and 3. In Tables 1 and 2, the least total CPU time to find the robust shortest path with HA and LA is bold typed, for each fixed n, d and k. The plots in Figures 5 and 6 show the average CPU times for $k = 2$ and $k = 3$, respectively, depending on the density of the network.

Tables 1 and 2 and Figures 5 and 6 show that preprocessing robust 0-persistent nodes can be more effective to solve the robust shortest path problem by HA or LA rather than without any preprocessing. Combining dynamic preprocessing with finding a robust shortest path was the most efficient method when HA was applied for $M = 1$ on the biggest networks ($n = 2000$, $d = 5$ and $k = 3$, or $n = 5000$, except for $d = 20$ and $k = 3$), as well as when LA was applied on most of the networks (except for $n = 500$ and $d = 20$). For all these cases, in spite of the preprocessing work demanded by Algorithm 2 being heavier than the required by the static version, $P_0^s < P_0^d$, the additional effort of the dynamic version leads to the detection of more robust 0-

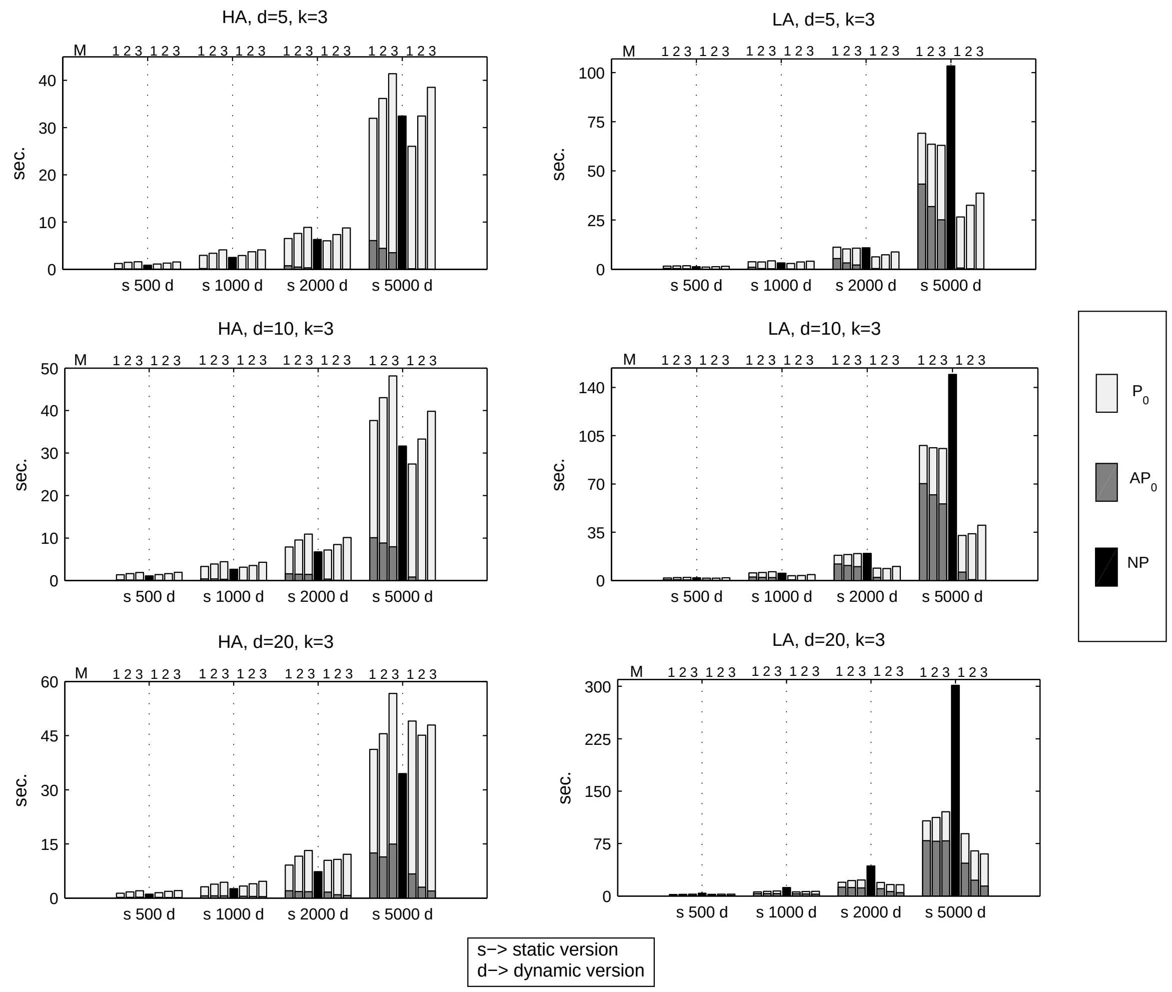

Figure 6: Average CPU times for preprocessing robust 0-persistent nodes and for algorithms HA and LA, with and without preprocessing, when $k = 3$

persistent nodes, $N_0^s < N_0^d$ – Table 3. This contributes for a more significant reduction of the network and consequently of the average CPU times when finding a robust shortest path after preprocessing, $AP_0^s > AP_0^d$. In conclusion, the dynamic version outperformed the static version. Besides, preprocessing with the dynamic search was also a better alternative than solving the problem without any preprocessing, $TP_0^d < NP$.

						HA					LA				
n	d	k	M	P_0^s	P_0^d	NP	AP_0^s	AP_0^d	TP_0^s	TP_0^d	NP	AP_0^s	AP_0^d	TP_0^s	TP_0^d
500	5	2	1	0.713	0.772	**0.596**	0.042	0.001	0.755	0.773	0.859	0.221	0.005	0.934	**0.777**
			2	0.948	0.986		0.024	0.000	0.972	0.986		0.114	0.003	1.062	0.989
		3	1	1.142	1.083	**0.857**	0.089	0.014	1.231	1.097	1.268	0.465	0.040	1.607	**1.123**
			2	1.384	1.308		0.059	0.003	1.443	1.311		0.304	0.010	1.688	1.318
			3	1.557	1.527		0.042	0.002	1.599	1.529		0.213	0.007	1.770	1.534
	10	2	1	0.877	1.060	**0.696**	0.089	0.010	0.966	1.070	1.763	0.528	0.033	1.405	**1.093**
			2	0.199	1.238		0.079	0.003	1.278	1.241		0.447	0.009	1.646	1.247
		3	1	1.201	1.318	**1.108**	0.155	0.080	1.356	1.398	1.948	0.675	0.396	1.876	**1.714**
			2	1.520	1.584		0.110	0.032	1.630	1.616		0.558	0.139	2.078	1.723
			3	1.777	1.915		0.101	0.020	1.878	1.935		0.517	0.065	2.294	1.980
	20	2	1	0.856	1.572	**0.772**	0.194	0.127	1.050	1.699	3.389	0.824	0.603	**1.680**	2.175
			2	1.127	1.630		0.183	0.089	1.310	1.719		0.789	0.371	1.916	2.001
		3	1	1.145	1.328	**1.053**	0.203	0.175	1.348	1.503	3.910	0.914	0.839	**2.059**	2.167
			2	1.481	1.723		0.198	0.157	1.679	1.880		0.883	0.761	2.364	2.484
			3	1.800	1.939		0.215	0.133	2.015	2.072		0.878	0.629	2.678	2.568
1000	5	2	1	1.869	1.974	**1.690**	0.152	0.002	2.021	1.976	2.410	0.878	0.020	2.747	**1.994**
			2	2.354	2.476		0.077	0.001	2.431	2.477		0.455	0.010	2.809	2.486
		3	1	2.783	2.873	**2.520**	0.156	0.020	2.939	2.893	3.192	1.002	0.107	3.785	**2.980**
			2	3.301	3.685		0.064	0.005	3.365	3.690		0.360	0.023	3.661	3.708
			3	4.055	4.084		0.037	0.002	4.092	4.086		0.199	0.013	4.254	4.097
	10	2	1	1.992	2.291	**1.792**	0.363	0.021	2.355	2.312	4.100	2.272	0.074	4.264	**2.365**
			2	2.634	2.753		0.325	0.008	2.959	2.761		2.033	0.023	4.667	2.776
		3	1	2.922	3.074	**2.646**	0.377	0.074	3.299	3.148	5.328	2.595	0.480	5.517	**3.554**
			2	3.543	3.532		0.361	0.022	3.904	3.554		2.279	0.110	5.822	3.642
			3	4.136	4.274		0.307	0.011	4.443	4.285		2.181	0.050	6.317	4.324
	20	2	1	2.083	2.737	**1.844**	0.480	0.138	2.563	2.875	8.579	2.897	0.833	4.980	3.570
			2	2.629	3.140		0.428	0.062	3.057	3.202		2.738	0.358	5.367	**3.498**
		3	1	2.547	2.845	**2.594**	0.586	0.488	3.133	3.333	12.061	3.391	3.051	5.938	**5.896**
			2	3.257	3.517		0.586	0.436	3.843	3.953		3.391	2.746	6.648	6.263
			3	3.787	4.223		0.580	0.411	4.367	4.634		3.358	2.524	7.145	6.747

Table 1: Average CPU time results for preprocessing robust 0-persistent nodes, $n \in \{500, 1000\}$

						HA					LA				
n	d	k	M	P_0^s	P_0^d	NP	AP_0^s	AP_0^d	TP_0^s	TP_0^d	NP	AP_0^s	AP_0^d	TP_0^s	TP_0^d
2000	5	2	1	4.744	4.872	**4.837**	0.267	0.007	5.011	4.879	7.522	1.807	0.045	6.551	**4.917**
			2	6.094	6.384		0.083	0.003	6.177	6.387		0.541	0.016	6.635	6.400
		3	1	5.745	5.977	6.297	0.748	0.054	6.493	**6.031**	10.922	5.475	0.323	11.220	**6.300**
			2	7.150	7.353		0.455	0.009	7.605	7.362		3.206	0.049	10.356	7.402
			3	8.559	8.748		0.297	0.002	8.856	8.750		2.194	0.034	10.753	8.782
	10	2	1	4.315	4.632	**4.634**	0.763	0.009	5.078	4.641	9.164	5.160	0.102	9.475	**4.734**
			2	5.637	5.883		0.599	0.005	6.236	5.888		3.948	0.031	9.585	5.914
		3	1	6.313	6.846	**6.757**	1.595	0.330	7.908	7.176	19.705	11.980	2.196	18.293	9.042
			2	8.047	8.431		1.509	0.047	9.556	8.478		10.839	0.234	18.886	**8.665**
			3	9.474	10.094		1.431	0.015	10.905	10.109		10.032	0.083	19.506	10.177
	20	2	1	4.823	5.845	**5.086**	1.950	0.127	6.773	5.972	33.345	12.860	0.833	17.683	**6.678**
			2	6.218	7.203		1.994	0.039	8.212	7.242		13.329	0.210	19.547	7.413
		3	1	7.140	8.809	**7.309**	2.007	1.629	9.147	10.438	42.829	12.605	10.543	19.745	19.352
			2	9.802	9.774		1.813	0.915	11.615	10.689		12.132	6.474	21.934	16.248
			3	11.392	11.421		1.767	0.715	13.159	12.136		11.531	4.808	22.923	**16.229**
5000	5	2	1	20.486	20.905	26.757	2.259	0.003	22.745	**20.908**	59.962	13.748	0.160	34.234	**21.065**
			2	26.391	26.770		0.845	0.006	27.236	26.776		4.952	0.032	31.343	26.802
		3	1	25.895	25.979	32.438	6.072	0.081	31.967	**26.060**	103.437	43.294	0.615	69.189	**26.594**
			2	31.760	32.382		4.414	0.056	36.174	32.438		31.870	0.198	63.630	32.580
			3	37.897	38.531		3.517	0.016	41.414	38.547		25.157	0.152	63.054	38.683
	10	2	1	21.449	21.797	26.601	9.967	0.014	31.416	**21.811**	134.070	53.750	0.187	75.199	**21.984**
			2	27.264	27.888		7.530	0.005	34.794	27.893		46.056	0.132	73.320	28.020
		3	1	27.601	26.594	31.671	10.070	0.843	37.671	**27.437**	149.398	70.250	6.142	97.851	**32.736**
			2	34.233	33.236		8.812	0.077	43.045	33.313		62.080	0.616	96.313	33.852
			3	40.223	39.827		7.930	0.018	48.153	39.845		55.514	0.224	95.737	40.051
	20	2	1	22.396	27.453	33.868	14.781	0.430	37.177	**27.883**	311.511	81.121	2.391	103.517	**29.844**
			2	29.453	33.095		14.009	0.069	43.462	33.164		82.884	0.527	112.337	33.622
		3	1	28.653	42.394	**34.468**	12.513	6.661	41.166	49.055	301.563	79.006	46.820	107.659	89.214
			2	34.135	42.061		11.397	3.018	45.532	45.079		78.269	22.580	112.404	64.641
			3	41.741	45.958		14.929	1.968	56.670	47.926		78.830	14.193	120.571	**60.151**

Table 2: Average CPU time results for preprocessing robust 0-persistent nodes, $n \in \{2000, 5000\}$

For each fixed n, d and k, the smaller the number of scenarios for testing (9), the less effort was required for computing the shortest path trees rooted at node 1. Hence, small values of M implied small preprocessing CPU times. This is valid for both the static and the dynamic approaches. The latter is always better than the first in detecting robust 0-persistent nodes, $N_0^s < N_0^d$, when M is fixed, as Table 3 shows. In general, the best value of M to consider in order to ensure that finding a robust shortest path with preprocessing is faster than solving the problem without preprocessing, must assure that $P_0 < NP$ and that the number of detected robust 0-persistent nodes is sufficient to reduce the CPU time which may not exceed $NP - P_0$. Tables 1 and 2 show that Algorithm 2 was more effective than its static version on such task when $M = 1$, except if $n = 500$, $d = 20$, $k \in \{2, 3\}$. When $M = 2$ or $M = 3$, the dynamic preprocessing combined with LA was the most efficient method in very few cases.

LA was always more sensitive to preprocessing than HA, and showed the most drastic reductions with respect to NP. This can be explained by the fact that removing nodes from the network allows to discard a considerable number of labels in LA, making easier the determination of an optimal solution. For HA, despite the fact that eliminating nodes reduces the effort on calculating reduced costs, preprocessing does not have so much impact, given that the search for a robust shortest path is more focused on selecting suitable deviation arcs and that can be done in few iterations without preprocessing [5].

			$n = 500$		$n = 1000$		$n = 2000$		$n = 5000$	
d	k	M	N_0^s	N_0^d	N_0^s	N_0^d	N_0^s	N_0^d	N_0^s	N_0^d
5	2	1	267	491	535	991	1518	1992	3247	4990
		2	361	495	714	994	1788	1995	4110	4994
	3	1	130	410	516	881	764	1730	1477	4646
		2	222	479	748	972	1126	1963	2193	4966
		3	279	493	834	989	1336	1992	2633	4994
10	2	1	149	430	221	903	911	1943	1260	4939
		2	196	483	290	974	1144	1990	1788	4993
	3	1	65	170	161	666	106	1250	353	3782
		2	120	324	236	871	188	1806	661	4724
		3	151	389	286	936	264	1925	900	4915
20	2	1	19	103	113	607	8	1662	119	4404
		2	34	201	146	776	14	1862	208	4806
	3	1	2	16	0	52	57	266	60	1383
		2	4	44	0	111	138	710	108	2713
		3	5	97	1	152	179	963	155	3248

Table 3: Average number of robust 0-persistent nodes

The number of detected robust 0-persistent nodes is high for the networks with the lowest densities ($d \in \{5, 10\}$), particularly for Algorithm 2 – Table 3. Moreover, when n, d and M are fixed, less nodes tend to be detected when k increases, since N_0^s and N_0^d also decrease. Globally, Figures 5 and 6 show that HA or LA have similar performances for the lowest densities ($d \in \{5, 10\}$). Moreover, LA is much more sensitive to the dynamic preprocessing than to the static preprocessing for all the densities, $|NP - TP_0^s| < |NP - TP_0^d|$.

6 Conclusions

In this work new techniques were developed to identify robust 1-persistent, and 0-persistent, arcs and nodes of the network. These techniques are dynamic versions of the preprocessing strategies presented in [6], because the tests they involve are updated as new paths are computed. The dynamic techniques were exemplified and its improvement towards the former versions was empirically tested on random instances, when combined with the labeling and hybrid algorithms introduced in [5].

The performed experiments revealed that, in general, the dynamic approach is the best choice for preprocessing robust 0-persistent nodes. Besides, LA was always more efficient after preprocessing than with no preprocessing at all. The same only happened with HA for networks with a large number of nodes using the dynamic preprocessing when $M = 1$, even for the cases for which the static approach was not efficient.

The improvement of the dynamic method, when compared to the static version in terms of the number of detected robust 0-persistent nodes ranged between 11% and 3600%. In general this reduction was also more demanding in terms of the CPU times. Nevertheless, for most of the cases the results showed that the total CPU time for solving the problem was still better when using the dynamic, rather than the static approach. The algorithms HA and LA after preprocessing with the dynamic method also outperformed the static version for almost all the cases. The maximum CPU time reduction was of 71%, when using LA, and of 31% when using HA. The biggest problems, in networks with 5000 nodes, 100 000 arcs and 3 scenarios, were solved in less than 10 seconds by HA and in less than 50 seconds by LA, after preprocessing.

Acknowledgments This work has been partially supported by the Portuguese Foundation for Science and Technology under project grants PEst-OE/ EEI/UI308/2014 and 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] P. Bille, A. Pagh, and R. Pagh. Fast evaluation of union-intersection expressions. Technical report, IT University of Copenhagen, Denmark, 2007.

[3] D. Catanzaro, M. Labbé, and M. Salazar-Neumann. Reduction approaches for robust shortest path problems. *Computers & Operations Research*, 38:1610–1619, 2011.

[4] O. E. Karasan, M. C. Pinar, and H. Yaman. The robust shortest path problem with interval data. Technical report, Bilkent University, Ankara, Turkey, 2001.

[5] M. Pascoal and M. Resende. Minmax regret robust shortest path problem in a finite multi-scenario model. *Applied Mathematics and Computation*, 241:88–111, 2014.

[6] M. Pascoal and M. Resende. Reducing the minmax regret robust shortest path problem with finite multi-scenarios. In P. Bourguignon, R. Jeltsch, A. Pinto, and M. Viana, editors, *CIM Series in Mathematical Sciences: Dynamics, Games and Science III.* Springer-Verlag, to appear, 2014.

[7] G. Yu and J. Yang. On the robust shortest path problem. *Computers Operations Research*, 25:457–468, 1998.