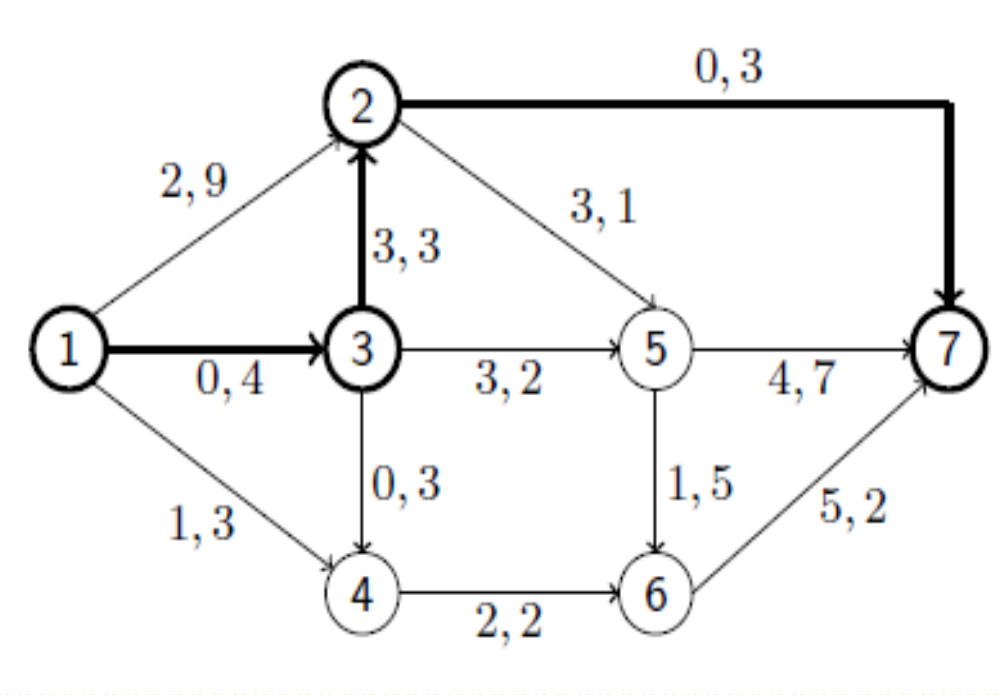


Marisa Cristina Marquês Neto de Matos Resende

THE ROBUST SHORTEST PATH PROBLEM WITH DISCRETE DATA

Tese de Doutoramento do Programa Inter-Universitário de Doutoramento em Matemática, orientada pela Professora Doutora Marta Margarida Braz Pascoal e apresentada ao Departamento de Matemática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Fevereiro de 2015

· U C ·

UNIVERSIDADE DE COIMBRA

Marisa Cristina Marquês Neto de Matos Resende

The robust shortest path problem with discrete data

Dissertação apresentada à Faculdade de Ciências e Tecnologia da Universidade de Coimbra, para a obtenção do grau de Doutor em Matemática na especialidade de Matemática Aplicada

Coimbra

2015

Acknowledgments

First of all, I would like to thank my advisor, Professor Marta Pascoal, for the motivation and useful hints while making my research. My recognition to her dedication, care and interest for preparing this thesis and for improving my skills and learnings on discrete optimization. I appreciate all the corrections and suggestions.

To the professors of the courses I attended in the first year of the PhD, I thank the academic preparation.

To the Department of Mathematics of the University of Coimbra and to the Institute for Systems Engineering and Computers at Coimbra (INESCC), I thank the working conditions for developing my work.

To the Foundation of Science and Technology, I thank the economic support during these last four years with grant SFRH/BD/51169/2010.

Finally, my acknowledgments are dedicated to all the people who contributed for my emotional support along the PhD. To my parents and my closest friends a sincere word of gratitude.

Resumo

Problemas de otimização são frequentemente utilizados para modelar fenómenos reais, sendo, deste modo, um instrumento de apoio à decisão. Tais modelos dependem de parâmetros que podem ser afetados pela incerteza. Na prática, é bastante comum que estes parâmetros assumam um conjunto de cenários possíveis, cujos valores podem variar num dado intervalo ou ser discretos. Uma forma de lidar com a incerteza, denominada otimização robusta, tem como objetivo minimizar o pior caso que possa ocorrer para todos os cenários. Esta tese considera o problema do caminho robusto mais curto com o mínimo desvio máximo de custo numa rede com um número finito de cenários. O problema consiste na determinação de um caminho entre dois nós, com o mínimo custo de robustez, isto é, com o mínimo desvio máximo de custo com respeito ao caminho mais curto em cada cenário. Na literatura, têm sido desenvolvidos predominantemente métodos de resolução do problema anterior, ou de algumas das suas variantes, quando são considerados intervalos de custo. Contudo, para o caso discreto, o tema não tem sido explorado significativamente.

Após serem demonstrados resultados fundamentais relativamente às soluções ótimas do problema do caminho robusto mais curto, são introduzidos três novos métodos exatos. Um deles é um método de rotulação, enquanto os outros dois têm por base a enumeração de caminhos limitados superiormente em termos de custo. Um destes algoritmos, denominado híbrido, utiliza uma técnica de desvio de caminhos para eliminar caminhos que o outro método de enumeração determina, por meio da aplicação de uma regra eliminatória utilizada no algoritmo de rotulação. Resultados computacionais em redes aleatórias mostram que as versões rotulação e híbrida são os algoritmos mais eficientes.

A redução da rede é outro dos tópicos tratados, com vista a simplificar a busca de um caminho robusto mais curto. Neste contexto, são desenvolvidas técnicas de pré-processamento para identificar arcos contidos em qualquer solução ótima, e nós que não estão contidos em nenhuma solução ótima. Os métodos seguem dois tipos de abordagem, estática e dinâmica. A primeira fixa o limite inferior dos custos, enquanto a segunda atualiza aqueles valores, de acordo com o mínimo custo de robustez dos caminhos que vão sendo obtidos. Nestas condições,

os conjuntos de arcos e de nós identificados pela última destas estratégias contêm os conjuntos identificados pelo método estático. Testes computacionais em redes geradas aleatoriamente mostram que as regras de pré-processamento são eficazes para os nós, se o número de cenários usados pelas condições testadas for limitado. Além disto, determinar um caminho robusto mais curto após pré-processamento dinâmico dos nós foi mais eficiente do que resolver o problema após o pré-processamento estático ou mesmo sem pré-processamento, em alguns casos.

O último tema aborda a reotimização do caminho robusto mais curto, após eliminação ou inclusão de cenários ou de arcos na rede inicial. Esta consiste na resolução do problema na rede modificada, assumindo que a solução ótima original, bem como os custos dos caminhos mais curtos e o valor ótimo iniciais são conhecidos. Inicialmente, são deduzidas condições que permitem verificar se a solução ótima se mantém. Quando estas não são satisfeitas, é desenvolvido um algoritmo para calcular a nova solução num conjunto de caminhos específico previamente determinado. O método de procura tem por base a construção de uma árvore de caminhos, que começa por incluir os sub-caminhos da solução ótima original que podem ser estendidos a caminhos potencialmente ótimos. Em cada passo, é adicionado um arco a cada caminho na árvore e as regras de extensão são aplicadas, de acordo com o mínimo custo de robustez conhecido na rede modificada. A aplicação dos algoritmos de reotimização é ilustrada.

Palavras-Chave: otimização em redes, cenários discretos, caminho robusto mais curto, pré-processamento, reotimização.

Abstract

Optimization problems are often used to model real phenomena and thus aid decision making. Such models depend on parameters that may be affected by uncertainty. In practice, it is quite common that these parameters are known to assume a given set of possible scenarios, which can range within an interval or be a discrete set of values. A way to handle uncertainty, called robust optimization, aims at minimizing the worst case that can happen for all scenarios. This thesis considers the minmax regret robust shortest path problem in a network, with a finite set of scenarios. The goal of this problem is to determine a path between two nodes, with the minimum robustness cost, that is, with the minimum maximum deviation cost with respect to the shortest path in each scenario. In the literature, methods to solve the latter problem or some of its variants have been mainly developed when each cost ranges within a given interval. However, for the discrete case, the subject has not been significantly explored.

After proving fundamental results concerning optimal solutions of the robust shortest path problem, three new exact methods to solve it are introduced. One is a labeling method, whereas the other two are based on an upper-bounded ranking of paths. One of these methods, denominated hybrid, uses a deviation technique that allows to skip some of the paths determined by the other ranking method, through the application of a pruning rule used in the labeling algorithm. Computational results in random networks reveal that the labeling and the hybrid versions are the most efficient algorithms.

The reduction of the network in order to simplify the search for a robust shortest path is also addressed. In this context, preprocessing techniques for identifying arcs that belong to all optimal solutions, and nodes that do not belong to any optimal solution, are developed. The methods follow two types of approach, static and dynamic. The first fixes the cost lower-bounds, while, the second, updates them according with the least robustness cost of the computed paths. Under these conditions, the sets of arcs or nodes identified by the latter strategy contain the sets identified by the static method. Computational results in randomly generated networks show that the preprocessing rules are effective for nodes, if the number of scenarios used in the test conditions is limited. Besides, determining a robust shortest path after the dynamic

preprocessing of nodes has shown to be more effective than solving the problem after the static preprocessing or even without preprocessing for some cases.

The reoptimization of the robust shortest path, after deleting or inserting scenarios or arcs in the initial network, is the last approached subject. It consists in solving the problem in the modified network, assuming that the original optimal solution as well as the shortest path costs and the optimal value in the initial network are known. First, conditions to verify if the optimal solution remains the same are deduced. When these do not hold, an algorithm is developed in order to calculate the new solution in a specific set of paths previously determined. The method is based on the construction of a paths tree, which initially includes the sub-paths of the original optimal solution that can be extended and produce potentially optimal paths. At each step, one arc is added to each path in the tree and the extension rules are applied, according with the least robustness cost known in the transformed network. The application of the reoptimization algorithms is illustrated.

Keywords: network optimization, discrete scenarios, robust shortest path, preprocessing, reoptimization.

Contents

Introduction

Optimization problems have long been used to model reality and, thus, to aid decision making. Traditionally, these problems have been treated in a deterministic manner, which means that the parameters involved in their description are assumed to be known and well determined. In reality, however, these values are often subjective, inaccurate, subject to changes, or sometimes unknown, making difficult the task of choosing reliable models. One approach for assigning plausible value(s) to each model parameter, considers the model nature and the possible variations of their parameters before optimizing the problem. In this context, the data uncertainty may be addressed by two types of optimization methods: stochastic and robust. The former is applied when probability laws may describe scenarios for the parameter values. However, in practice, it can be difficult to know an exact distribution for the data and to enumerate scenarios that reproduce them. Hence, the robust optimization arises as a common alternative by assuming scenarios with deterministic values.

In combinatorial optimization, looking for robust solutions is usually related with determining a solution that is good taking into account uncertainty, more precisely, a good solution for the generality of the possible scenarios. Thus, the goal is, generally, to find the solution that behaves the best in the worst case. To determine the worst case scenario, the robustness strategy must take into account what is affected by uncertainty, namely, whether it is the feasibility of the solution or the objective function value. In particular, the works [7, 43] cover problems handling both latter aspects.

When uncertainty affects the feasibility of a solution, robust optimization is focused on seeking for a solution that is feasible for any realization taken by the unknown coefficients of the uncertainty set, which is centered around the nominal values of uncertain parameters. This methodology comes widely from the mathematical programming formulation over convex sets in the literature [6]. Since the middle of the 90's, two models were particularly developed: discrete and interval data models. In the former case, there is a finite number of possible cost scenarios, whereas in the latter the costs can range within given intervals.

When uncertainty affects the optimality of a solution, and, therefore, the objective function

to be considered, several kinds of robustness measures are adopted in order to establish a criterion to solve the problem [46]. When paths are evaluated in terms of robustness, three types of criteria can be distinguished. The first is classical and results from the decision theory, which considers the minimization of an objective function that expresses the worst case in terms of cost or the regret cost [27]. The second methodology comes from the resolution of the mathematical programming formulation, by considering cost intervals around the nominal values of the objective function coefficients and a parameter of adjustment, in order to control the number of coefficients that deviate from their nominal value [7]. The third approach is based on multicriteria analysis, where the decision problem is commonly defined using a set of solutions, a discrete set of criteria and an aggregation model of these criteria. In this context, the robustness analysis is based on the choice of a suitable evaluation vector associated with each solution and on the definition of an aggregation model for the obtained evaluation vectors [45].

This thesis approaches the classical criteria of robustness, which considers the minimization of one of two possible objective functions. One that represents the maximum cost among all scenarios, known as the absolute robustness cost, and another that represents the maximum deviation cost with respect to the best cost over all scenarios, known as the relative robustness cost. When the purpose is to find a path between a given pair of nodes in a network, this leads to the absolute robust shortest path problem, in the first case, and to the minmax regret robust shortest path problem, in the second. Some works have addressed these problems for interval data, however the literature on the discrete case is rather scarce.

Most of the research on the robust shortest path problem with interval data is based on a discretization of the intervals into two particular scenarios, which result from considering only the interval lower and upper-limits. This idea was first applied to the case of acyclic networks, in 2001, by Karasan, Pinar and Yaman [26]. This work also introduced rules to reduce the network, after the determination of arcs that do not belong to any optimal solution. In 2004-2005, Montemanni and Gambardella [31, 33, 32] proposed new algorithms for the interval data robust shortest path problem, based on the idea used in [26]. More recently, Catanzaro, Labbé and Salazar-Neumann [13] developed enhanced pre-processing techniques to reduce the network before finding a robust shortest path, for any type of network.

When arc costs may have a finite set of realizations, i.e., a finite set of scenarios, Dias and Clímaco [16] considered that only a finite set of cost realizations, i.e., a finite set of scenarios is known. Given that not all information is available, those authors explored the problem from a multicriteria point of view, and proposed the determination of a set of non-dominated paths, that is, a set of paths that are not worse than any other for all scenarios. Perny and Spanjaard [42] also used the concept of dominance to develop specific rules in state space graphs as an axiomatic approach to robustness. In 1992, Murthy and Her [35] showed that an

optimal solution of the absolute robust shortest path problem with finite scenarios must be a non-dominated path. As a result, they developed a labeling algorithm to solve the problem, which combines dominance tests on the labels with pruning techniques, to discard some labels which cannot lead to an optimal solution. The relative version of the problem was introduced only in 1998, by Yu and Yang [50], who designed a dynamic programming strategy to solve it. The methods are pseudo-polynomial in time but were shown not to be effective for problems with large cost upper-bounds or a large number of scenarios. Moreover, it was also shown that the problem is strongly NP-hard when the number of scenarios is unbounded. To overcome such difficulties, an exact method was conceived specifically for layered networks and heuristics were developed to compute an approximate optimal solution in general. In 2010, Bruni and Guerriero [11] proposed several heuristics and evaluation functions to guide the search performed by Murthy and Her's algorithm. Empirical tests have shown that this enhanced the original version of the method. To our knowledge, these are the only works that address the robust shortest path problem when the arc costs assume a discrete set of scenarios.

The book by Kouvelis and Yu [27] provides a complete state-of-the-art on robust optimization for combinatorial problems in general, up to the end of last century. Additionally, more recently, survey papers have also been published on this topic, [12, 22, 23].

As mentioned above, this work focuses the robust shortest path problem with discrete data. It has three main goals, to propose exact methods to solve the problem efficiently, to introduce conditions and procedures that enable the identification of arcs or nodes of the network which can be deleted before actually solving the problem, and to study techniques to reoptimize an optimal path if the set of network arcs or the set of network scenarios is modified. The remainder of this work is organized into five chapters. The first is dedicated to the introduction of notation and the definition of the minmax regret robust shortest path problem. Then, preliminary results to be used in the following are presented. Namely, the existence of solutions for this problem is analyzed and some of their properties are derived, concerning uniqueness and simplicity.

In Chapter 2, three exact algorithms are presented to determine an optimal simple path between a pair of nodes of a network. These methods were introduced in [39], the first of these algorithms is a variant of the labeling approach proposed in [35], adapted to the minmax regret objective function. The cost lower and upper-bounds are used similarly. The second algorithm ranks simple paths by non-decreasing order of cost. A cost upper-bound is used to limit the ranking. The search can be restricted along the process, according to the costs of the optimal path candidates. The third algorithm is a hybrid version of the two previous methods. This algorithm ranks simple paths, however the cost bounds imposed for the first method are applied. This allows to discard useless solutions at an early stage, and, therefore, to skip some of the paths ranked in the second algorithm. The time complexity orders of the proposed algorithms

are deduced. An example of the application of the introduced methods is provided and, then, computational results over random instances are presented.

Preprocessing techniques are derived in Chapter 3. Their purpose is to reduce the network into a subnetwork where an optimal solution can still be identified. Inspired by the results in [13], sufficient conditions are established to detect arcs that surely belong to all optimal solutions and nodes that do not belong to any of them. The preprocessing rules are implemented following two approaches: static and dynamic. The first, the static version introduced in [40], sets and fixes a cost lower-bound involved on a test condition that is constant along the algorithm. The second, the dynamic version introduced in [41], aims to enhance the former strategy by applying the same type of conditions, but updating the cost lower-bounds and the arcs or nodes scanned along the algorithm, as paths are computed. The two types of algorithms are described and their time computational complexity orders are determined. The arcs and nodes identified by the preprocessing procedures as useless are eliminated from the original network and a robust shortest path is found on the reduced model by applying the labeling and the hybrid algorithms presented in Chapter 2. The impact of the static and dynamic procedures is evaluated by means of empirical tests on randomly generated instances, comparing the running times for solving the problem with and without preprocessing. The obtained results are analyzed and discussed.

In Chapter 4, the reoptimization of the robust shortest path problem is addressed, assuming that the network changes by the elimination or the inclusion of scenarios or arcs. It is assumed that the original optimal solution, its robustness cost and the shortest path costs are known in the initial network. In a first step, conditions for verifying whether the former optimal path does not change after the modifications are established. When these conditions do not hold, the new optimal solution needs to be computed, being possible to restrict the search for the latter to a subset of paths in the modified network. With this purpose, a labeling method that constructs a paths tree is designed. The method starts with the sub-paths of the original optimal solution, except the optimal path itself, which are defined in the modified network and can be extended to an optimal solution. This requirement is evaluated using cost lower-bounds as in the pruning technique of the labeling and the hybrid approaches of Chapter 2. The same rule is applied whenever an arc is added to the paths tree. Besides, when any of those arcs ends in the terminal node of the network, it is verified whether the obtained path belongs to the search set previously defined and if its robustness cost in the modified network improves the least attained. The established results and the associate algorithms are exemplified.

The last chapter is devoted to the presentation of a summary of the main results of the developed work and of final conclusions. Some open questions and possible research directions are highlighted.

Chapter 1

Preliminaries

In this chapter some preliminary concepts are presented. Namely, notation is introduced, the robust shortest path problem is defined and some of its properties are discussed.

1.1 Problem definition and notation

A finite multi-scenario model is represented by $G = G(V, A, S)$, where G is a directed graph with a set of nodes $V = \{1, \ldots, n\}$, a set of m arcs $A \subseteq \{(i, j) : i, j \in V \text{ and } i \neq j\}$ and a finite set of scenarios $S = \{s_1, \ldots, s_k\}$, $k > 1$. When indexing scenarios, U_k is used to denote the set $\{1, \ldots, k\}$. The density or average degree of G is denoted by d, which is given by $d = m/n$.

For each arc $(i, j) \in A$, i and j are named the tail and the head node, respectively, and $c_{ij}^{s_u}(G) \in \mathbb{R}$ represents its cost under scenario s_u in G, $u \in U_k$. It is assumed that the graph contains no parallel arcs, nor self-loops.

Let A^* and S^* be nonempty sets of arcs and scenarios, respectively. Then, $G^-_{A^*}$ and $G^-_{S^*}$ denote the subgraphs of G with set of arcs $A \backslash A^*$ and set of scenarios $S \backslash S^*$, respectively. Analogously, $G^+_{A^*}$ and $G^+_{S^*}$ denote the extensions of G with set of arcs $A \cup A^*$ and set of scenarios $S \cup S^*$, respectively. To simplify notation, when $A^* = \{(i, j)\}$ and $S^* = \{s_u\}$, $u \in U_k$, the representations $G^-_{(i,j)}$ and $G^-_{s_u}$, for the subgraphs $G^-_{A^*}$ and $G^-_{S^*}$, and the representations $G^+_{(i,j)}$ and $G^+_{s_u}$, for the extensions $G^+_{A^*}$ and $G^+_{S^*}$, are adopted.

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

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

with $v_1 = i$, $v_r = j$ and where $v_l \in V$, for $l = 2, \ldots, r-1$, and $(v_l, v_{l+1}) \in A$, for $l = 1, \ldots, r-1$. 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 final 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, 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 simple if all its nodes are different.

The cost of a path p in G under scenario s_u, $u \in U_k$, is defined by

$$c_G^{s_u}(p) = \sum_{(i,j)\in A(p)} c_{ij}^{s_u}(G).$$

With no loss of generality, 1 and n denote the origin and the destination nodes of the graph G, respectively. The set of all (i,j)-paths in G is represented by $P_{ij}(G)$, $i,j \in V$.

Let $p_{ij}^{l,s_u}(G)$ represent the l-th shortest (i,j)-path of G, $i,j \in V$, in scenario s_u, $u \in U_k$. In order to simplify the notation, $p^{l,s_u}(G)$ is used to denote the l-th shortest $(1,n)$-path of G in scenario s_u, i.e. $p_{1n}^{l,s_u}(G)$, and $LB_{ij}^{s_u}(G)$ is used to denote the cost of the shortest (i,j)-path of G in scenario s_u, i.e. $c_G^{s_u}(p_{ij}^{1,s_u}(G))$.

For each scenario s_u, $u \in U_k$, the trees of the paths $p_{1i}^{1,s_u}(G)$ and of the paths $p_{in}^{1,s_u}(G)$, for any $i \in V$, are represented by $\mathcal{T}_1^{s_u}(G)$ and $\mathcal{T}_n^{s_u}(G)$, respectively, and 1 and n are called the roots of these trees.

The minmax regret robust shortest path problem in G corresponds to determining a path in $P_{1n}(G)$ with the least maximum robust deviation, i.e. satisfying

$$\arg \min_{p\in P_{1n}(G)} RC_G(p), \tag{1.1}$$

where $RC_G(p)$ is the robustness cost in G of a path $p \in P_{1n}(G)$, defined by

$$RC_G(p) := \max_{u\in U_k} RD_G^{s_u}(p),$$

where $RD_G^{s_u}(p)$ represents the robust deviation in G of a path $p \in P_{1n}(G)$ under scenario s_u, $u \in U_k$, defined by

$$RD_G^{s_u}(p) := c_G^{s_u}(p) - LB_{1n}^{s_u}(G).$$

Any optimal solution is called a robust shortest path of G. The set of scenarios indices in which $RC_G(p)$ occurs is denoted by $U_G(p) = \{\arg\max_{u\in U_k} RD_G^{s_u}(p)\}$.

The idea behind minimizing the maximum robust deviation is to find a $(1,n)$-path with the best deviation cost in all scenarios, with respect to the shortest $(1,n)$-path in each one. A problem that resembles this one is the minmax shortest path problem [35]. The latter is an absolute version of problem (1.1), for which the objective function to minimize is $\max_{u\in U_k}\{c_G^{s_u}(p)\}$. Both problems have the same optimal solution if the cost $LB_{1n}^{s_u}(G)$ is constant for any $u \in U_k$.

For the theoretical results in the remainder of this chapter and in Chapters 2 and 3, G is used to represent a general network. For particular examples, other notations can be considered. In Chapter 4, other variants of G are represented according to the introduced modifications.

1.2 Properties of the optimal solutions

In this section, the existence of a robust shortest path is analyzed. Then, some elementary properties of the optimal solutions in G are deduced, concerning uniqueness and cyclic nature. This last property will be a requirement to be satisfied when searching for an optimal solution when developing algorithms.

To start, two preliminary results are settled. The first is related with the sign of the robustness cost of any $(1, n)$-path of G.

Proposition 1.1. *For every $p \in P_{1n}(G)$, $RC_G(p) \geq 0$.*

Proof. Let $p \in P_{1n}(G)$. By definition of shortest $(1, n)$-path in scenario s_u, $LB_{1n}^{s_u}(G) \leq c_G^{s_u}(p)$, for any $u \in U_k$. The result is then an immediate consequence of the definition of robustness cost of a $(1, n)$-path. □

This proposition assures the existence of a minimum among the robustness costs of the paths of $P_{1n}(G)$, given that this set is non-empty and finite by assumption. Therefore, the existence of a solution for the robust shortest path problem is always assured. Moreover, any path in $P_{1n}(G)$ with null robustness cost must be a robust shortest path of G.

The second result is a particular consequence of the previous proposition and of the problem definition, because it characterizes a robust shortest path of G with null robustness cost as a shortest $(1, n)$-path of G for all scenarios.

Corollary 1.2. *A path p is a shortest $(1, n)$-path of G in every scenario of S, if and only if p is a robust shortest path of G, satisfying $RC_G(p) = 0$.*

The network $G_1 = G_1(V, A, \{1, 2\})$, depicted in Figure 1.1, shows that a robust shortest path may not be unique. In fact, for this example, $p^{1,1}(G_1) = \langle 1, 2, 4\rangle$ and $p^{1,2}(G_1) = \langle 1, 3, 4\rangle$. Under these conditions, the two $(1, 4)$-paths $p^{1,1}(G_1)$ and $q = \langle 1, 2, 3, 4\rangle$ are both robust shortest paths of G_1, as they have the minimum robustness cost 2 $\big(U_{G_1}(p^{1,1}(G_1)) = \{2\}$ and $U_{G_1}(q) = \{1, 2\}\big)$.

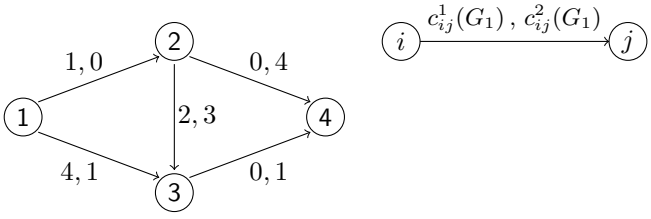


Figure 1.1: Network G_1

In order to develop algorithms that compute a $(1, n)$-path with the minimum robustness cost in G, other properties must be established. An important result concerns the cyclic nature

of an optimal solution. In fact, any robust shortest path on an acyclic network is naturally simple. Nevertheless, in a network containing cycles, there may exist robust shortest paths including cycles as well. In fact, let G_2 be the network represented in Figure 1.2. In this case, the shortest $(1,4)$-paths of G_1 for scenarios 1 and 2 are the same for G_2 $\big(p^{1,1}(G_2) = \langle 1,2,4\rangle$ and $p^{1,2}(G_2) = \langle 1,3,4\rangle\big)$. Hence, the simple paths $p^{1,1}(G_2) = \langle 1,2,4\rangle$ and $q = \langle 1,2,3,4\rangle$ are still optimal solutions in G_2, with the same robustness cost 2. However, path $q' = \langle 1,2,1,2,4\rangle$ is a new robust shortest path of G_2, containing the cycle $\langle 1,2,1\rangle$, given that $RC_{G_2}(q') = 2$ $\big(U_{G_2}(q') = \{1,2\}\big)$.

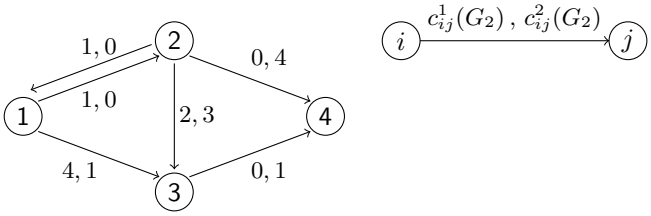


Figure 1.2: Network G_2

Although there may exist robust shortest paths containing cycles, Yu and Yang [50] proved the existence of a simple optimal solution for networks with non-negative arc costs for all scenarios. This result is still valid for networks without cycles with negative cost in any scenario, as shown in Proposition 1.3.

Proposition 1.3. *Let G be a network without cycles with negative cost in any scenario, then there exists a simple robust shortest path in G.*

Proof. Let $p \in P_{1n}(G)$. In case p is simple, consider $p' = p$, otherwise, consider p' as the $(1,n)$-path resultant from p by deleting all its cycles. Under these conditions, p' is a simple $(1,n)$-path. Since, by assumption, all the cycles in G must have a non-negative cost for all scenarios,

$$c_G^{s_u}(p') \leq c_G^{s_u}(p)\,, \text{ for any } u \in U_k.$$

Hence, by definition of robustness cost of a $(1,n)$-path,

$$RC_G(p') \leq RC_G(p).$$

Consequently, there exists a simple path $p' \in P_{1n}(G)$ such that $RC_G(p') = \min_{p \in P_{1n}(G)} RC_G(p)$, i.e. there exists a simple robust shortest path in G. □

As a consequence of this property, it is assumed from now on that G is a network without cycles with negative cost in any scenario. Under these conditions, it is enough to solve the robust shortest path problem by scanning the simple paths of $P_{1n}(G)$ only.

Chapter 2

Algorithms for the robust shortest path problem

As mentioned in the introductory chapter, robust optimization has greatly developed in the recent years, as a way to handle uncertainty that deals with the variability of the parameters of the problem. The present chapter is dedicated to the discrete case of the robust shortest path problem, in particular. Namely, algorithms are developed and illustrated for this problem, and their computational complexities in terms of the performed operations are deduced. Finally, their performance is evaluated by means of a set of tests on random instances.

2.1 Introduction

Similarly to what generally happens with the shortest path problem, the methods introduced in this chapter compute a simple robust shortest path. These methods were introduced in [39] and explore two traditional approaches to deal with the shortest path problem, and more concretely with its multiobjective version: labeling and ranking.

The first approach can be seen as an extension of the shortest path label correcting algorithms, like Bellman-Ford-Moore's algorithm [5, 19, 34], to the case with several cost functions, each one associated with one scenario. The application is not straightforward, given that the present problem does not follow the Bellman's principle of optimality [5]. Therefore, the growth of the search tree formed by the algorithm has to be controlled by means of additional pruning rules. Works that propose labeling algorithms with a variety of strategies, in the context of constrained, or multiobjective, shortest path problems can be found in [10, 35, 25, 28, 48].

The second approach is based on ranking paths by non-decreasing order of their cost according to a chosen scenario. Because it has been proven that the robust shortest path problem has a simple optimal solution (Proposition 1.3), the ranking can be limited to simple paths.

One of the dangers of such an approach is related with the number of simple paths that may have to be calculated before an optimal solution is found. In order to cope with this issue, the robustness costs of the ranked paths are used to obtain an upper-bound to halt the ranking.

A third approach to the robust shortest path problem is also proposed. This results from a combination of the two previous methods, in the sense that it uses an algorithm for ranking shortest simple paths, complemented with pruning rules tuned to skip useless paths.

2.2 Labeling approach

The method presented in this section for computing a robust shortest path of G is inspired on the labeling algorithm proposed by Murthy and Her [35] for the minmax shortest path problem. This problem consists of determining a $(1, n)$-path with the minimum maximum cost over all scenarios and it does not satisfy Bellman's principle of optimality [5], as said above. However, an optimal path must be non-dominated, which can be assured in case all its sub-paths are non-dominated as well. Here, the concept of dominance is applied to the costs of the paths in $P_{1i}(G)$, $i \in V$, for all scenarios. With this knowledge, Murthy and Her create labels for each of the nodes of G and apply dominance tests to select only the labels associated with the sub-paths that could be contained in an optimal path. The method is complemented by rules for pruning useless labels. One of them is based on the use of cost lower-bounds for the paths of $P_{in}(G)$, $i \in V$, in each scenario. The other results from the Lagrangian relaxation of the subproblem of the linear programming formulation obtained when the previous bounds are fixed.

Like the minmax shortest path problem, the robust shortest path problem does not satisfy Bellman's principle of optimality either. Thus, the algorithm described in the following for the latter problem has an approach similar to the method of Murthy and Her, because of the dominance tests and the pruning rule that uses cost lower-bounds to extend sub-paths to node n. The main modifications concern the adaptation of the labels and upper-bounds to the new objective function. Moreover, the dominance tests for the labels associated with the computed $(1, n)$-paths are skipped. Along the algorithm, a search tree rooted at node 1 is constructed, labeled by the costs vector for each path in that tree. Some notation is now introduced.

Let $z_G(p_{1i}) = (z^1_G(p_{1i}), \ldots, z^k_G(p_{1i}))$ denote a label associated with a path $p_{1i} \in P_{1i}(G)$, $i \in V$. Each u-th component of a label is related with a cost for p_{1i} in scenario s_u, $u \in U_k$. By default, it is considered that node 1 is a $(1, 1)$-path of G represented by $\langle 1 \rangle$. Because the objective function to evaluate $(1, n)$-paths depends on the robust deviations for all scenarios, in order to simplify intermediate calculations, the label associated with node 1 is given by

$$z_G(\langle 1 \rangle) = (-LB^{s_1}_{1n}(G), \ldots, -LB^{s_k}_{1n}(G)).$$

Along the algorithm, the labels can be recursively obtained when an arc is added to the final node of a path previously selected. Specifically, given $p_{1i} \in P_{1i}(G)$ and its label, $z_G(p_{1i})$, $i \in V$, the label associated with $p_{1j} = p_{1i} \diamond \langle i, j \rangle$, can be obtained, for any $(i, j) \in A$, according to the formula

$$z_G(p_{1j}) = (z_G^1(p_{1i}) + c_{ij}^{s_1}(G), \ldots, z_G^k(p_{1i}) + c_{ij}^{s_k}(G)).$$

With the above initialization,

$$z_G(p_{1n}) = (RD_G^{s_1}(p_{1n}), \ldots, RD_G^{s_k}(p_{1n})),$$

is the vector of robust deviations of a $(1, n)$-path p_{1n}. Then, by definition, the robustness cost of p_{1n} in G can be determined by

$$RC_G(p_{1n}) = \max_{u \in U_k} z_G^u(p_{1n}).$$

Under these conditions, a robust shortest path of G is found among the paths of $P_{1n}(G)$ with labels having the least maximum component. This result is stated in the following lemma.

Lemma 2.1. *Let $p_{1n} \in P_{1n}(G)$. Then, p_{1n} is a robust shortest path of G if and only if*

$$\max_{u \in U_k} z_G^u(p_{1n}) \leq \max_{u \in U_k} z_G^u(p'_{1n}),$$

for any $p'_{1n} \in P_{1n}(G)$.

Any $(1, n)$-path p_{1n} can be eliminated as potentially optimal, when its label $z_G(p_{1n})$ does not satisfy the condition of Lemma 2.1. Nevertheless, extending all the paths of $P_{1i}(G)$, $i \in V \backslash \{n\}$, to all possible $(1, n)$-paths can be computationally demanding. Therefore, two pruning techniques will be derived with the aim of discarding in an early stage of the algorithm, paths that cannot be part of a simple robust shortest path, which allows to reduce the total number of labels that have to be stored along the calculations as well. For the first reduction rule, new concepts related with the dominance of the generated labels are given.

Definition 2.2. *Let $p_{1i}, p'_{1i} \in P_{1i}(G)$, $i \in V$. Then, the label associated with p_{1i}, $z_G(p_{1i})$, dominates the label associated with p'_{1i}, $z_G(p'_{1i})$, if*

$$z_G^1(p_{1i}) \leq z_G^1(p'_{1i}), \ldots, z_G^k(p_{1i}) \leq z_G^k(p'_{1i})$$

and at least one of the inequalities is strict.

Let $Z_G(P_{1i})$ denote the set of labels for all the paths of $P_{1i}(G)$, $i \in V$.

Definition 2.3. *A label of $Z_G(P_{1i})$ is efficient (or non dominated) if there is no other label in $Z_G(P_{1i})$ that dominates it.*

In [35], the dominance tests are applied to the labels in each set $Z_G(P_{1i})$, $i \in V$. For the method presented here, the tests are skipped for the labels in $Z_G(P_{1n})$, which are only selected when the inequality in Lemma 2.1 is strict. In this way, computational effort can be spared.

Two aspects must be taken into account for each set $Z_G(P_{1i})$, $i \in V\backslash\{n\}$. One is the dominance between its labels and the other is their equivalence, which happens when they have equal components. Proposition 2.4 allows to discard any $(1,i)$-path p_{1i} with a label dominated by, or equivalent to, a label of some other $(1,i)$-path p'_{1i}. In fact, in either case, it is shown that the $(1,n)$-path that results from extending p_{1i} is never better than the same extension of p'_{1i}.

Proposition 2.4. *Let $p_{1i}, p'_{1i} \in P_{1i}(G)$, $q \in P_{in}(G)$, $i \in V\backslash\{n\}$, and $p_{1n} = p_{1i} \diamond q$, $p'_{1n} = p'_{1i} \diamond q$.*

1. *If $z_G(p_{1i})$ is dominated by $z_G(p'_{1i})$, then $RC_G(p'_{1n}) \leq RC_G(p_{1n})$.*

2. *If $z_G(p_{1i})$ is equivalent to $z_G(p'_{1i})$, then $RC_G(p'_{1n}) = RC_G(p_{1n})$.*

Proof.

1. If $z_G(p_{1i})$ is dominated by $z_G(p'_{1i})$, then,

$$z^u_G(p'_{1n}) = z^u_G(p'_{1i}) + c^{s_u}_G(q) \leq z^u_G(p_{1i}) + c^{s_u}_G(q) = z^u_G(p_{1n}), \text{ for any } u \in U_k,$$

   with

$$z^{u'}_G(p'_{1n}) < z^{u'}_G(p_{1n}), \text{ for some } u' \in U_k.$$

   Consequently,

$$\max_{u \in U_k} z^u_G(p'_{1n}) \leq \max_{u \in U_k} z^u_G(p_{1n}),$$

   and the result follows from the definition of robustness cost of a $(1,n)$-path.

2. If $z_G(p_{1i})$ is equivalent to $z_G(p'_{1i})$, then,

$$z^u_G(p'_{1i}) = z^u_G(p_{1i}), \text{ for any } u \in U_k,$$

   and the reasoning applied to point 1. allows to obtain

$$\max_{u \in U_k} z^u_G(p'_{1n}) = \max_{u \in U_k} z^u_G(p_{1n}).$$

   The result follows by definition of robustness cost of a $(1,n)$-path.

□

As a consequence of the last result, the paths of $P_{1i}(G)$, $i \in V\backslash\{n\}$, with labels not dominated by, or equivalent to, another in $Z_G(P_{1i})$ can be considered for possible extension. Since it is intended to restrict the search to simple $(1,n)$-paths, a first in first out (FIFO) policy for managing the paths under evaluation can be adopted. This means that breadth-search is used to build the search-tree and that when equivalent labels occur in each $Z_G(P_{1i})$, $i \in V\backslash\{n\}$, only the associate $(1,i)$-path which is generated first is stored. To show this result, a representation for the subsets of $P_{1i}(G)$ associated with equivalent labels is introduced in the following. Specifically, given $p_{1i} \in P_{1i}(G)$,

$$[p_{1i}]_G^{eq} = \{p'_{1i} \in P_{1i}(G) : z_G(p'_{1i}) \text{ is equivalent to } z_G(p_{1i})\}$$

defines the set of paths of $P_{1i}(G)$ with labels equivalent to $z_G(p_{1i})$. It can be easily checked that the equivalence between the labels of $Z_G(P_{1i})$ is an equivalence relation, because the equalities between the components of the labels are, trivially, reflexive, symmetric and transitive. Under these conditions, $[p_{1i}]_G^{eq}$ is an equivalence class of paths with respect to the equivalence relation between the associate labels. Proposition 2.5 shows that, when paths are managed in a FIFO policy, it is always possible to obtain a simple robust shortest path of G with each of its sub-paths being first generated, when other paths exist in the associate equivalence class.

Proposition 2.5. *Assume that the set of paths for scanning is managed as a FIFO list. Then, there exists a simple robust shortest path q, such that each of its $(1,i)$-sub-paths, q_{1i}, is the first generated in $[q_{1i}]_G^{eq}$, $i \in V(q)\backslash\{n\}$.*

Proof. If there exists a simple robust shortest path, q, of G, such that all of its $(1,i)$-sub-paths, q_{1i}, $i \in V(q)\backslash\{n\}$, satisfy $\left|[q_{1i}]_G^{eq}\right| = 1$, the result is immediate. It remains to prove that q_{1i} is the first generated in $[q_{1i}]_G^{eq}$, when $\left|[q_{1i}]_G^{eq}\right| > 1$, $i \in V(q)\backslash\{n\}$. By contradiction, assume that no simple robust shortest path q exists under such conditions. Let p^* be a simple robust shortest path and $j \in V(p^*)$ be its node closest to node 1 such that $\left|[p^*_{1j}]_G^{eq}\right| > 1$, with p^*_{1j} the $(1,j)$-sub-path of p^*. Suppose $p'_{1j} \neq p^*_{1j}$ is the first path of $[p^*_{1j}]_G^{eq}$ to be generated. Denote by p^*_{jn} the (j,n)-path in p^*. Then, $p'_{1j} \diamond p^*_{jn}$ is a $(1,n)$-path, such that

$$RC_G(p'_{1j} \diamond p^*_{jn}) = RC_G(p^*).$$

Hence, $p'_{1j} \diamond p^*_{jn}$ is a robust shortest path of G and p'_{1j} was the first path generated in $[p'_{1j}]_G^{eq} = [p^*_{1j}]_G^{eq}$. By assumption, $p'_{1j} \diamond p^*_{jn}$ should contain a cycle. Let x be the first repeated node in $p'_{1j} \diamond p^*_{jn}$, and $p'_{1x} \diamond p^*_{xn}$ be the simple $(1,n)$-path obtained from $p'_{1j} \diamond p^*_{jn}$ after removal of that cycle. Here p'_{1x} and p^*_{xn} correspond to p'_{1j}'s sub-path from 1 to x and p^*_{jn}'s sub-path from x to n, respectively. Again,

$$RC_G(p'_{1x} \diamond p^*_{xn}) = RC_G(p'_{1j} \diamond p^*_{jn}),$$

and $p'_{1x} \diamond p^*_{xn}$ is a simple robust shortest path of G. By hypothesis, and because paths are managed as a FIFO list, all the $(1,i)$-sub-paths, p'_{1i}, of p'_{1j} are the first to be generated in $[p'_{1i}]^{eq}_G$, $i \in V(p'_{1j})$. So the same happens for the $(1,i)$-sub-paths, p'_{1i}, of p'_{1x}, $i \in V(p'_{1x})$, because $V(p'_{1x}) \subseteq V(p'_{1j})$. For $i \in V(p^*_{xn})\backslash\{n\}$, if all the $(1,i)$-sub-paths, $p'_{1x} \diamond p^*_{xi}$, of $p'_{1x} \diamond p^*_{xn}$ are the first to be generated in $[p'_{1x} \diamond p^*_{xi}]^{eq}_G$, then the result is proven by considering $q = p'_{1x} \diamond p^*_{xn}$. Otherwise, because p^*_{xn} has less nodes than p^*_{jn}, the reasoning can be repeated a finite number of times leading to the existence of a simple path q under the stated conditions, which contradicts the assumption.

□

From now on, it will be assumed that the paths to be extended are treated in a FIFO manner. Otherwise, it should be verified whether a selected path contains a cycle or not.

A second pruning rule for the paths of $P_{1i}(G)$, $i \in V\backslash\{n\}$, is inferred in Proposition 2.6. This property is based on a bounding condition satisfied by every sub-path of a $(1,n)$-path.

Proposition 2.6. *Let $p \in P_{1n}(G)$ and p_{1i} be a $(1,i)$-sub-path of p, $i \in V\backslash\{n\}$. Then,*

$$\max_{u \in U_k}\{z^u_G(p_{1i}) + LB^{s_u}_{in}(G)\} \leq RC_G(p).$$

Proof. Let p_{1i} be a $(1,i)$-path, $i \in V\backslash\{n\}$, contained in p. Then,

$$c^{s_u}_G(p) \geq c^{s_u}_G(p_{1i} \diamond p^{1,s_u}_{in}(G)) = c^{s_u}_G(p_{1i}) + LB^{s_u}_{in}(G)\,, \text{ for any } u \in U_k$$

or, equivalently,

$$RD^{s_u}_G(p) \geq z^u_G(p_{1i}) + LB^{s_u}_{in}(G)\,, \text{ for any } u \in U_k,$$

and the result follows from the definition of the robustness cost of a $(1,n)$-path in G. □

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

$$\max_{u \in U_k}\{z^u_G(p_{1i}) + LB^{s_u}_{in}(G)\} \geq UB \tag{2.1}$$

holds with a strict inequality, then the $(1,i)$-path p_{1i} cannot be part of any optimal solution. In case of an equality, p_{1i} can be part of an optimal solution with robustness cost UB. Nevertheless, taking into account that a candidate path with the same robustness cost is already known, p_{1i} and $z_G(p_{1i})$ can be discarded in both cases. If (2.1) is satisfied with a strict inequality, this pruning rule is equivalent to the first one proposed in [35].

The value UB is initialized with the best robustness cost of the shortest $(1, n)$-paths for each scenario, keeping in mind that calculating their costs is fundamental to start the algorithm. Hence, UB is initialized with

$$\min_{u \in U_k} RC_G(p^{1,s_u}(G)) = \min_{u \in U_k} \max_{u' \in U_k} RD_G^{s_{u'}}(p^{1,s_u}(G)). \tag{2.2}$$

This value is then updated as new labels for $(1, n)$-paths are computed.

The structure of the labeling algorithm for finding a robust shortest path is described in the following.

Global algorithmic structure To start with, the computation of the trees $\mathcal{T}_n^{s_u}(G)$ and of the associate costs $LB_{in}^{s_u}(G)$, $i \in V \backslash \{n\}$, are necessary, for each $u \in U_k$. Any shortest path tree algorithm can be applied with such purpose [1]. Then, the optimal cost upper-bound UB is initialized with (2.2). In order to do that, calculating the deviation costs for the shortest $(1, n)$-paths of G over all scenarios is required. Since some of them can be the shortest for more than one scenario, the computation of their robustness costs can be avoided by using a list Q with only the distinct shortest paths. The first candidate is the path of Q with the least robustness cost.

A variable $RCaux$ stores the robustness cost of a $(1, n)$-path after its label has been calculated. It updates UB in case it improves the least robustness cost found so far. The variable *sol* represents any potentially optimal $(1, n)$-path.

A list X collects the paths of $P_{1i}(G)$ to be scanned and list Z_i stores the labels associated with the paths in X, $i \in V \backslash \{n\}$. List X is managed under a FIFO policy. When implementing the code, in order to save space memory, only the last node of each generated path is stored, rather than the whole path. In this way, the optimal solution can be retrieved at the end of the algorithm, by tracing back the nodes up to node 1.

When selecting $p_{1i} \in X$ for extension, a $(1, j)$-path $p_{1j} = p_{1i} \diamond \langle i, j \rangle$, $(i, j) \in A$, $j \in V \backslash \{n\}$, is not discarded when its label, $z_G(p_{1j})$, is not dominated by, or equivalent to, another label in Z_j and it does not satisfy (2.1). Then, all the labels in list Z_j dominated by $z_G(p_{1j})$ are removed, as well as the corresponding $(1, j)$-paths in X. Afterwards, path p_{1j} and $z_G(p_{1j})$ are inserted in lists X and Z_j, respectively.

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

Algorithm 1: Labeling approach for finding a robust shortest path of G

$Q \leftarrow \emptyset$;
for $u \in U_k$ **do**
  Compute $\mathcal{T}_n^{s_u}(G)$; $Q \leftarrow Q \cup \{p^{1,s_u}(G)\}$;
  for $i = 1, \ldots, n-1$ **do** $LB_{in}^{s_u}(G) \leftarrow c_G^{s_u}(p_{in}^{1,s_u}(G))$;
$UB \leftarrow \min\{RC_G(q) : q \in Q\}$;
$sol \leftarrow q$ such that $q \in Q$ and $RC_G(q) = UB$;
$z_G(\langle 1 \rangle) \leftarrow (-LB_{1n}^{s_1}(G), \ldots, -LB_{1n}^{s_k}(G))$; $X \leftarrow \{\langle 1 \rangle\}$; $Z_1 \leftarrow \{z_G(\langle 1 \rangle)\}$;
for $i = 2, \ldots, n-1$ **do** $Z_i \leftarrow \emptyset$;
while $X \neq \emptyset$ **do**
  $p_{1i} \leftarrow$ first path in X; $X \leftarrow X - \{p_{1i}\}$;
  for $(i,j) \in A$ **do**
    $p_{1j} \leftarrow p_{1i} \diamond \langle i,j \rangle$;
    $z_G(p_{1j}) \leftarrow (z_G^1(p_{1i}) + c_{ij}^{s_1}(G), \ldots, z_G^k(p_{1i}) + c_{ij}^{s_k}(G))$;
    if $j = n$ **then**
      $RCaux \leftarrow \max\{z_G^u(p_{1j}) : u \in U_k\}$;
      if $RCaux < UB$ **then** $UB \leftarrow RCaux$; $sol \leftarrow p_{1j}$;
    else if $z_G(p_{1j})$ *is not dominated by, or equivalent to, any label in* Z_j **and** $\max_{u \in U_k}\{z_G^u(p_{1j}) + LB_{jn}^{s_u}(G)\} < UB$ **then**
      Delete from Z_j all the labels dominated by $z_j(p_{1j})$;
      Delete from X the $(1,j)$-paths associated with the labels deleted from Z_j;
      $X \leftarrow X \cup \{p_{1j}\}$; $Z_j \leftarrow Z_j \cup \{z_G(p_{1j})\}$;
return sol;

Computational time complexity order In order to determine the worst case computational complexity of Algorithm 1, some auxiliary procedures are analyzed.

1. **Determination of a tree $\mathcal{T}_n^{s_u}(G)$, for some $u \in U_k$, and of the associate costs $c_G^{s_{u'}}(p_{in}^{1,s_u}(G))$, $i \in V$, for any $u' \in U_k$**: 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 [21]. Computing the costs for all scenarios has $\mathcal{O}(kn)$ in both cases, so this step has $\mathcal{O}(m+kn)$ time for acyclic networks and $\mathcal{O}(m + n\log n + kn)$ time for general networks.

2. **Calculation of $RC_G(p^{1,s_u}(G))$, for some $u \in U_k$, given $c_G^{s_{u'}}(p^{1,s_u}(G))$, for any $u' \in U_k$**: The k robust deviations, $RD_G^{s_{u'}}(p^{1,s_u}(G))$, $u' \in U_k$, are obtained in $\mathcal{O}(k)$ time and their minimum can be found with $\mathcal{O}(k)$ comparisons. Thus, the required work has $\mathcal{O}(k)$ time.

3. **Generation of a label given another**: The label of a path $p_{1j} = p_{1i} \diamond \langle i,j \rangle$, $(i,j) \in A$, is obtained from the label of p_{1i}, $i \in V$, by adding the costs of arc (i,j), for all scenarios of S. Hence, this calculation is performed in $\mathcal{O}(k)$ time.

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

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

The second stage concerns computing the search tree of paths through the generation, scanning and pruning of the associate labels. Let W denote the maximum number of paths in each set $P_{1i}(G)$, $i \in V\backslash\{n\}$, that are generated (a value dependent on the parameters n, m and k). Then, $W(n-1)$ is the maximum number of iterations of the **while** loop in line 9 of Algorithm 1, and each of them implies at most $n-1$ iterations of the **for** loop in line 11. In each of these iterations, the calculation of a new label is done in $\mathcal{O}(k)$ time, the dominance tests for the labels are performed in $\mathcal{O}(kW)$, and (2.1) is checked in $\mathcal{O}(1)$. Additionally, updating X, Z_i $i \in V\backslash\{n\}$, takes one operation, therefore, the second phase of Algorithm 1 has complexity of $O_2 = \mathcal{O}(kn^2W^2)$.

Therefore, Algorithm 1 has a time complexity of $\mathcal{O}(k^2n + kn^2W^2)$ for any type of network, since $\log n \ll n$ and $m < n^2$.

Example Let $G_3 = G_3(V, A, \{1, 2\})$ be the network depicted in Figure 2.1, and consider the application of Algorithm 1 for finding a robust shortest path in G_3.

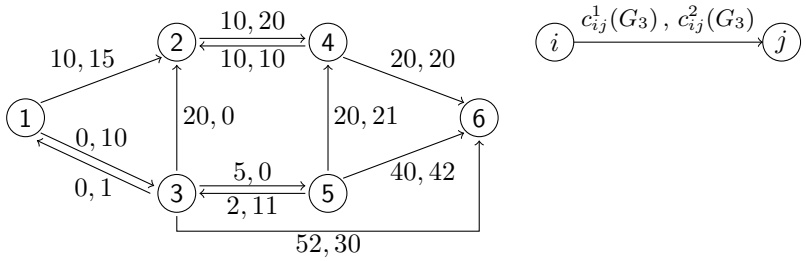
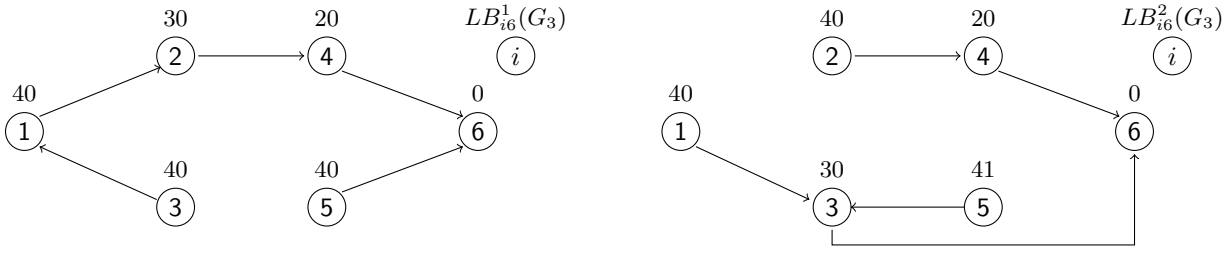


Figure 2.1: Network G_3

The plots in Figure 2.2 show the trees $\mathcal{T}_6^1(G_3)$ – Figure 2.2.(a) – and $\mathcal{T}_6^2(G_3)$ – Figure 2.2.(b). The values attached to each tree node i represent the cost of the $(i, 6)$-path in that tree,

$i = 1, \ldots, 6$.



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

Figure 2.2: Shortest path trees rooted at $n = 6$ in G_3

Initially, the elements of set Q are the shortest $(1, 6)$-paths of G_3 in scenarios 1 and 2, i.e. $p^{1,1}(G_3) = \langle 1, 2, 4, 6\rangle$, with $LB^1_{16}(G_3) = 40$, and $p^{1,2}(G_3) = \langle 1, 3, 6\rangle$, with $LB^2_{16}(G_3) = 40$. Because $c^1_{G_3}(p^{1,2}(G_3)) = 52 < c^2_{G_3}(p^{1,1}(G_3)) = 55$, $p^{1,2}(G_3)$ is the path in Q with the minimum robustness cost, 12, which allows to set initially

$$UB = 12 \quad \text{and} \quad sol = \langle 1, 3, 6\rangle.$$

Figure 2.3 shows the tree of paths that is obtained when applying Algorithm 1.

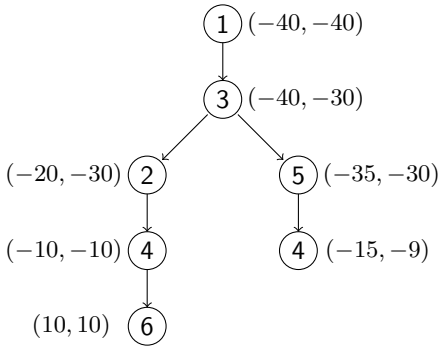


Figure 2.3: Search tree of paths for G_3 produced by Algorithm 1

The method starts by selecting the label

$$z_{G_3}(\langle 1\rangle) = (-LB^1_{16}(G_3), -LB^2_{16}(G_3)) = (-40, -40)$$

and including it in Z_1. From node 1, paths $\langle 1, 2\rangle$ and $\langle 1, 3\rangle$ are generated, with labels $(-30, -25)$ and $(-40, -30)$, respectively. However, path $\langle 1, 2\rangle$ is discarded, because it leads to $(1, 6)$-paths

with a robustness cost that does not improve UB, given that

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,2\rangle) + LB^u_{26}(G_3)\} = 15 > UB.$$

Moreover,

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,3\rangle) + LB^u_{36}(G_3)\} = 0 \leq UB,$$

which means that the extension of $\langle 1,3\rangle$ might be optimal. Then, $z_{G_3}(\langle 1,3\rangle) = (-40,-30)$ is inserted in Z_3. After that, paths $\langle 1,3,1\rangle$, $\langle 1,3,2\rangle$, $\langle 1,3,5\rangle$ and $\langle 1,3,6\rangle$ are created with labels $(-40,-29)$, $(-20,-30)$, $(-35,-30)$ and $(12,0)$, respectively. The first of them is dominated by label $(-40,-40)$ in Z_1, therefore, the former is eliminated together with its path. Nevertheless, because

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,3,2\rangle) + LB^u_{26}(G_3)\} = 10 \leq UB$$

and

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,3,5\rangle) + LB^u_{56}(G_3)\} = 11 \leq UB,$$

paths $\langle 1,3,2\rangle$ and $\langle 1,3,5\rangle$ are stored in X and the associate labels, $(-20,-30)$ and $(-35,-30)$, are included in Z_2 and Z_5, respectively. Path $\langle 1,3,6\rangle$ is discarded, because its robustness cost is not better than UB,

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,3,6\rangle)\} = UB.$$

After selecting the next path in X, $\langle 1,3,2\rangle$, path $\langle 1,3,2,4\rangle$ is created, with the label $(-10,-10)$. Because

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,3,2,4\rangle) + LB^u_{46}(G_3)\} = 10 \leq UB,$$

this path and its label are inserted in X and Z_4, respectively. Path $\langle 1,3,5\rangle$ is the next to be picked from X, and extended to paths $\langle 1,3,5,3\rangle$, $\langle 1,3,5,4\rangle$ and $\langle 1,3,5,6\rangle$, which have labels $(-33,-19)$, $(-15,-9)$ and $(5,12)$, respectively. The first of these labels is dominated by $(-40,-30)$ in Z_3, so it is deleted together with its path. Path $\langle 1,3,5,6\rangle$ and its label are not stored either, because

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,3,5,6\rangle)\} = UB.$$

The label $z_{G_3}(\langle 1,3,5,4\rangle) = (-15,-9)$ is not dominated by the label $(-10,-10)$ in Z_4, neither is this label dominated by $(-15,-9)$. In addition, the extensions of the path $\langle 1,3,5,4\rangle$ can produce $(1,6)$-paths with a robustness cost of at least

$$\max_{u \in U_2}\{z^u_{G_3}(\langle 1,3,5,4\rangle) + LB^u_{46}(G_3)\} = 11 \leq UB.$$

Thus, $\langle 1,3,5,4\rangle$ is inserted in X and its label is included in Z_4. The path associated with the other label in Z_4 is extended to $\langle 1,3,2,4,6\rangle$, which has a label of $(10,10)$ and a robustness cost of 10. Then, the following updates are performed

$$UB = 10 \quad \text{and} \quad sol = \langle 1,3,2,4,6\rangle.$$

Path $\langle 1,3,5,4\rangle$ remains to be extended, but the only possibility is to consider $\langle 1,3,5,4,6\rangle$, which has a label of $(5,11)$. Its robustness cost is 11, which is not better than UB. Hence, $\langle 1,3,2,4,6\rangle$ is the robust shortest path of G_3.

2.3 Ranking approach

This section presents an alternative strategy to the previous for determining a simple robust shortest path. It is based on ranking simple $(1,n)$-paths by non-decreasing order of cost under a fixed scenario until a previously set cost upper-bound is reached. This technique is inspired on the work of Dias and Clímaco [16], who considered the determination of a set of $(1,n)$-paths that are not dominated in terms of cost with respect to any scenario. With this goal, they adapted the bicriteria shortest path algorithm by Clímaco and Martins [14]. This strategy was particularly useful when continuous models were considered, after discretizing the cost intervals using simply their lower and upper-limits. For the robust shortest path problem with finite multi-scenarios, some adaptations to the previous method can be made, taking into account the new optimal values according to the number of scenarios involved. Namely, only simple $(1,n)$-paths have to be ranked and the update of the cost upper-bounds according to the least produced robustness costs can be done till an optimal path is found.

In the following, the algorithmic procedures are explained in detail. To start with, the next result provides an upper-bound on the robust shortest path cost under particular scenarios.

Proposition 2.7. *Let $q \in P_{1n}(G)$. If p is a robust shortest path of G, then*

$$c_G^{s_u}(p) \leq c_G^{s_u}(q)\,, \quad \textit{for any } u \in U_G(q).$$

Proof. Let $q \in P_{1n}(G)$, u be any element of $U_G(q)$, and p be a robust shortest path of G. By definition of $U_G(q)$ and of the robustness cost of a $(1,n)$-path,

$$RC_G(q) = RD_G^{s_u}(q).$$

By contradiction, assume that p satisfies $c_G^{s_u}(p) > c_G^{s_u}(q)$. Then, by definition of robust deviation and of robustness cost of a $(1,n)$-path, one deduces that

$$RC_G(p) \geq RD_G^{s_u}(p) > RD_G^{s_u}(q) = RC_G(q).$$

Consequently, p cannot be a robust shortest path of G, which contradicts the assumption. □

Proposition 2.7 allows to establish the scenario under which simple $(1,n)$-paths are ranked and the first associate cost upper-bound. In fact, once a $(1,n)$-path q is set as candidate for robust shortest path, a better candidate can be found by ranking $(1,n)$-paths from $p^{1,s_r}(G)$ to q, with $r \in U_G(q)$. Under this condition, $RD_G^{s_r}(q) = RC_G(q)$, and, consequently, $c_G^{s_r}(q) = LB_{1n}^{s_r}(G) + RC_G(q)$ is set as the first cost upper-bound of the ranking. For an imposed cost upper-bound associated with any robust shortest path candidate q, the search for an optimal solution must consider only the paths of the ranking with a cost under scenario s_r smaller than $c_G^{s_r}(q)$. In fact, the paths with that cost will have a robustness cost of at least $RC_G(q)$ and the goal of the algorithm is to find only one optimal solution. Whenever a path $p^{l,s_r}(G)$, $l \geq 1$, is found along the ranking, satisfying $RC_G(p^{l,s_r}(G)) < RC_G(q)$, the cost upper-bound can be improved. The next result shows this and that it is also possible to detect an optimal solution when $r \in U_G(p^{l,s_r}(G))$.

Proposition 2.8. *Let $q \in P_{1n}(G)\backslash\{p^{1,s_r}(G)\}$ and $r \in U_k$. Let $p^{l,s_r}(G) \neq q$, $l \geq 1$, satisfy:*

1. $RC_G(p^{l,s_r}(G)) < RC_G(q)$,
2. $RC_G(p^{l',s_r}(G)) \geq RC_G(q)$, $\forall l' : 1 \leq l' < l$.

Then, any robust shortest path of G is of the form $p^{\hat{l},s_r}(G)$, for some $\hat{l} \geq l$, and it satisfies

$$c_G^{s_r}(p^{\hat{l},s_r}(G)) \leq LB_{1n}^{s_r}(G) + RC_G(p^{l,s_r}(G)) < LB_{1n}^{s_r}(G) + RC_G(q). \tag{2.3}$$

Moreover, if $r \in U_G(p^{l,s_r}(G))$, then $p^{l,s_r}(G)$ is a robust shortest path as well.

Proof. Given $q \in P_{1n}(G)\backslash\{p^{1,s_r}(G)\}$ and $r \in U_k$, let $p^{l,s_r}(G) \neq q$, $l \geq 1$, be a path obtained when ranking in scenario s_r, satisfying conditions 1. and 2. Then, according to these inequalities, every robust shortest path of G must be a $(1,n)$-path of the form $p^{\hat{l},s_r}(G)$, for some $\hat{l} \geq l$, since otherwise, it does not have the minimum robustness cost in $P_{1n}(G)$. Hence,

$$RC_G(p^{\hat{l},s_r}(G)) \leq RC_G(p^{l,s_r}(G)),$$

and, therefore,

$$RD_G^{s_r}(p^{\hat{l},s_r}(G)) \leq RC_G(p^{l,s_r}(G)),$$

according to the definition of robustness cost. Consequently,

$$c_G^{s_r}(p^{\hat{l},s_r}(G)) \leq LB_{1n}^{s_r}(G) + RC_G(p^{l,s_r}(G)).$$

and, then, from condition 1., one concludes (2.3).

In addition to 1. and 2., let now be assumed that $r \in U_G(p^{l,s_r}(G))$. For this case, s_r is the scenario under which the robust deviation of $p^{l,s_r}(G)$ is maximum, which means that

$$RC_G(p^{l,s_r}(G)) = RD_G^{s_r}(p^{l,s_r}(G)).$$

Moreover, because $\hat{l} \geq l$,

$$RD_G^{s_r}(p^{l,s_r}(G)) \leq RD_G^{s_r}(p^{\hat{l},s_r}(G)) \leq RC_G(p^{\hat{l},s_r}(G)),$$

where the last inequality follows from the definition of robustness cost. Consequently,

$$RC_G(p^{l,s_r}(G)) \leq RC_G(p^{\hat{l},s_r}(G)).$$

Since, by assumption, $p^{\hat{l},s_r}(G)$ is a robust shortest path of G, one must have

$$RC_G(p^{l,s_r}(G)) = RC_G(p^{\hat{l},s_r}(G)),$$

and, therefore, $p^{l,s_r}(G)$ is a robust shortest path of G as well. □

Recalling that the goal is to find a simple robust shortest path, the last result applies when q is a simple path and only simple paths $p^{l,s}(G)$, $l \geq 1$, are ranked for scenario s_r. Under these assumptions, for each calculated simple path $p^{l,s_r}(G)$, two conditions are analyzed to determine how the ranking can be shortened. The first condition is related with the improvement of $RC_G(q)$. In fact, if $RC_G(p^{l,s_r}(G)) < RC_G(q)$, the cost upper-bound $LB_{1n}^{s_r}(G) + RC_G(q)$ can be decreased to $LB_{1n}^{s_r}(G) + RC_G(p^{l,s_r}(G))$. The second condition is checked if the first is satisfied, and it consists on the identification of the ranking scenario with one of the scenarios in which $RC_G(p^{l,s_r}(G))$ occurs. This is crucial to spare computational effort, because if $r \in U_G(p^{l,s_r}(G))$, the search can halt, given that it can be concluded that $p^{l,s_r}(G)$ is an optimal solution.

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

Analogously to Algorithm 1, the upper-bound for the least robustness cost is denoted by UB and initialized with the least robustness cost for the shortest $(1, n)$-paths of G over all scenarios. Another important issue concerns the choice of the scenario for the ranking. For that, let $p^{1,s_{u'}}(G)$ be a shortest $(1, n)$-path with robustness cost equal to the initial UB, for some $u' \in \{1, \ldots, k\}$. Under these conditions, $p^{1,s_{u'}}(G)$ is a first candidate optimal solution, and, without loss of generality, the smallest r in $U_G(p^{1,s_{u'}}(G))$ will define the ranking scenario index. That is

$$r = \min\{u \in \{1, \ldots, k\} : RD_G^{s_u}(p^{1,s_{u'}}(G)) = RC_G(p^{1,s_{u'}}(G))\}. \tag{2.4}$$

Consequently, $c_G^{s_r}(p^{1,s_{u'}}(G))$ is set as the first cost upper-bound for the ranking in scenario s_r.

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

Global algorithmic structure The preliminary procedures for this approach are similar to Algorithm 1 and the variables Q, UB, $RCaux$ and sol represent the same. Another variable stores the cost upper-bound for the ranking, $Cmax$. The ranking scenario s_r is initialized according to (2.4).

Several algorithms can be applied to rank simple paths in general networks, for instance [29, 30, 38, 49]. For acyclic networks, unconstrained ranking algorithms, which are generally more efficient, can be used, like [18, 30].

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

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

Algorithm 2: Ranking approach for finding a robust shortest path of G

```
1  Q ← ∅;
2  for u ∈ U_k do
3      Compute p^{1,s_u}(G); Q ← Q ∪ {p^{1,s_u}(G)};
4      LB_{1n}^{s_u}(G) ← c_G^{s_u}(p^{1,s_u}(G));
5  UB ← min{RC_G(q) : q ∈ Q};
6  sol ← q such that q ∈ Q and RC_G(q) = UB;
7  r ← min{u ∈ U_k : RD_G^{s_u}(sol) = UB}; Cmax ← c_G^{s_r}(sol); l ← 2;
8  while p^{l,s_r}(G) exists do
9      Compute p^{l,s_r}(G);
10     if c_G^{s_r}(p^{l,s_r}(G)) ≥ Cmax then break;
11     if p^{l,s_r}(G) ∉ Q then
12         RCaux ← RC_G(p^{l,s_r}(G));
13         if RCaux < UB then
14             UB ← RCaux; sol ← p^{l,s_r}(G);
15             if RD_G^{s_r}(p^{l,s_r}(G)) = UB then break;
16             Cmax ← LB_{1n}^{s_r}(G) + UB;
17     l ← l + 1;
18 return sol;
```

Computational time complexity order In order to determine the worst case 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 are still valid for the first phase of Algorithm 2. Thus, such procedures are performed in $O_1^a = \mathcal{O}(km + k^2n)$ time for acyclic networks and in $O_1^c = \mathcal{O}(k(m + n\log n) + k^2n)$ time for the general case. According to (2.4), the choice of the ranking scenario is done in $\mathcal{O}(k)$ time, which does not affect the previous bounds.

If L simple paths are ranked in scenario s_r, the time is of $\mathcal{O}(m + n\log n + L\log L)$ for acyclic networks, using Eppstein's algorithm [18], and of $\mathcal{O}(Ln(m+n\log n))$ in the general case, applying Yen's algorithm or one of its variants [29, 38, 49]. Parameter L depends on n, m and k, and cannot be known in advance. Because the shortest path $p^{1,s_r}(G)$ was previously computed, the second phase ranks the remaining $L-1$ simple paths in $\mathcal{O}(n\log n + L\log L)$ for acyclic networks and in $\mathcal{O}(Ln(m + n\log n))$ for general networks. In the worst case, the robustness costs of all the ranked paths, besides $p^{1,s_r}(G)$, have to be determined. The cost of each of those paths in a given scenario can be computed in $\mathcal{O}(n)$ 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 $O_2^a = \mathcal{O}(L(\log L + kn) + n\log n)$ time for acyclic networks and in $O_2^c = \mathcal{O}(Ln(k + m + n\log n))$ time for general networks.

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

Example Consider the application of Algorithm 2 to the network G_3, depicted in Figure 2.1, in order to find a simple robust shortest path. The initial upper-bound UB and the first candidate for the optimal solution *sol* are determined as in Algorithm 1. That is,

$$UB = 12 \text{ and } sol = p^{1,2}(G_3) = \langle 1, 3, 6\rangle.$$

Since $U_{G_3}(\langle 1, 3, 6\rangle) = \{1\}$, the simple paths will be ranked in scenario 1, with a first cost upper-bound set to

$$Cmax = c^1_{G_3}(\langle 1, 3, 6\rangle) = 52.$$

Table 2.1 summarizes the steps of Algorithm 2.

l	$p^{l,1}(G_3)$	$c^1_{G_3}(p^{l,1}(G_3))$	$c^2_{G_3}(p^{l,1}(G_3))$	Updates
1	$\langle 1,2,4,6\rangle$	40	55	$UB \leftarrow 12$; $sol \leftarrow p^{1,2}(G_3)$; $Cmax \leftarrow 52$
2	$\langle 1,3,5,6\rangle$	45	52	$RCaux \leftarrow 12$
3	$\langle 1,3,5,4,6\rangle$	45	51	$UB = RCaux \leftarrow 11 < 12$; $sol \leftarrow p^{3,1}(G_3)$; $RD^1_{G_3}(p^{3,1}(G_3)) \neq 11$; $Cmax \leftarrow 51$
4	$\langle 1,3,2,4,6\rangle$	50	50	$UB = RCaux \leftarrow 10 < 11$; $sol \leftarrow p^{4,1}(G_3)$; $RD^1_{G_3}(p^{4,1}(G_3)) = 10$; Stop

Table 2.1: Simulation of Algorithm 2 for G_3

The computation of the second shortest $(1,6)$-path in scenario 1, $p^{2,1}(G_3) = \langle 1,3,5,6\rangle$, with a robustness cost of 12, does not improve UB, which demands the calculation of path $p^{3,1}(G_3) = \langle 1,3,5,4,6\rangle$. This new path has a robustness cost smaller than the previous, 11, which allows to update UB, the potential optimal solution sol and the cost upper-bound $Cmax$ with

$$UB = 11 \ ; \ sol = p^{3,1}(G_3) \ \text{ and } Cmax = c^1_{G_3}(p^{3,1}(G_3)) = 51.$$

Since the maximum robust deviation of $p^{3,1}(G_3)$ does not occur in scenario 1, the next path in the ranking must be obtained. The robustness cost, 10, of such path, $p^{4,1}(G_3) = \langle 1,3,2,4,6\rangle$, is the least obtained so far, updating

$$UB = 10,$$

and, moreover, it occurs under scenario 1. Consequently, the algorithm halts, returning

$$sol = p^{4,1}(G_3) = \langle 1,3,2,4,6\rangle$$

as the optimal solution.

2.4 Hybrid approach

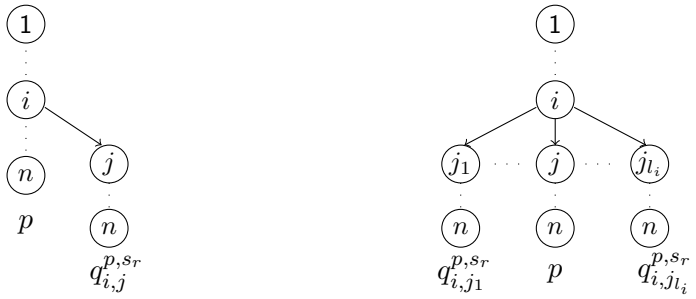
The method presented in this section results from the combination of Algorithm 2 and some pruning techniques used in Algorithm 1. In order to apply these pruning rules in the broadest possible way, a specific ranking algorithm will be used, based on the deviation algorithm MPS, introduced by Martins, Pascoal and Santos [30]. The idea is to skip some useless paths of the ranking for a set cost upper-bound in order to determine a robust shortest path in fewer iterations than with Algorithm 2. For completeness, first, the MPS method is very briefly reviewed. After that, the deviation algorithm used here is described and the rules applied to discard useless paths are presented. Unless otherwise stated, it is assumed that the ranking is done with respect to a given scenario s_r, $r \in U_k$.

Let $p \in P_{1n}(G)$, $i \in V(p)$, and p_{1i} denote the $(1,i)$-sub-path of p. The idea behind deviation algorithms for ranking paths, or simple paths, is to generate l-th shortest $(1,n)$-path candidates,

$l > 1$, as paths that coincide with p along p_{1i} and that deviate from p exactly at node i. Because the aim of such methods is to rank $(1, n)$-paths by order of cost, p_{1i} is extended with $(i, j) \in A$ and the shortest (j, n)-path for scenario s_r, according to Figure 2.4.(a). Hence, the generated paths have the form

$$q_{i,j}^{p,s_r} = p_{1i} \diamond \langle i, j \rangle \diamond p_{jn}^{1,s_r}(G), \quad (i, j) \in A. \tag{2.5}$$

In this case, p is called the father of $q_{i,j}^{p,s_r}$. Additionally, i and (i, j) are denominated the deviation node and the deviation arc of $q_{i,j}^{p,s_r}$, respectively, and this path is said to be a deviation of p. When $i = 1$, p_{1i} reduces to the initial node, 1. By convenience, it is considered that the father of $p^{1,s_r}(G)$ is not defined and that 1 is its deviation node.



(a) Generation of a path (b) Ranking deviation paths

Figure 2.4: Deviation method

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

$$\bar{c}_{ij}^{s_r}(G) = LB_{jn}^{s_r}(G) - LB_{in}^{s_r}(G) + c_{ij}^{s_r}(G).$$

The reduced cost of a path $p \in P_{1n}(G)$ in scenario s_r is then given by

$$\bar{c}_G^{s_r}(p) = \sum_{(i,j) \in A(p)} \bar{c}_{ij}^{s_r}(G).$$

Now, because $\bar{c}_{ij}^{s_r}(G) = 0$ for any $(i, j) \in \mathcal{T}_n^{s_r}(G)$, then $\bar{c}_G^{s_r}(p_{jn}^{1,s_r}(G)) = 0$, for any $j \in V$. Hence,

$$\bar{c}_G^{s_r}(q_{i,j}^{p,s_r}) = \bar{c}_G^{s_r}(p_{1i}) + \bar{c}_{ij}^{s_r}(G), \ (i, j) \in A,$$

and, therefore, the shortest path with form (2.5) with respect to scenario s_r must contain the arc with tail node i with the minimum reduced cost.

Let $\mathcal{A}_G^{s_r}(i) = \{(i, j_1), \ldots, (i, j_{l_i})\}$ represent the set of arcs of G with tail node i sorted by non-decreasing order of the reduced costs with respect to scenario s_r, that is, such that

$$\bar{c}_{ij_1}^{s_r}(G) \leq \ldots \leq \bar{c}_{ij_{l_i}}^{s_r}(G).$$

Therefore,

$$\bar{c}_G^{s_r}(q_{i,j_1}^{p,s_r}) \leq \ldots \leq \bar{c}_G^{s_r}(q_{i,j_{l_i}}^{p,s_r}),$$

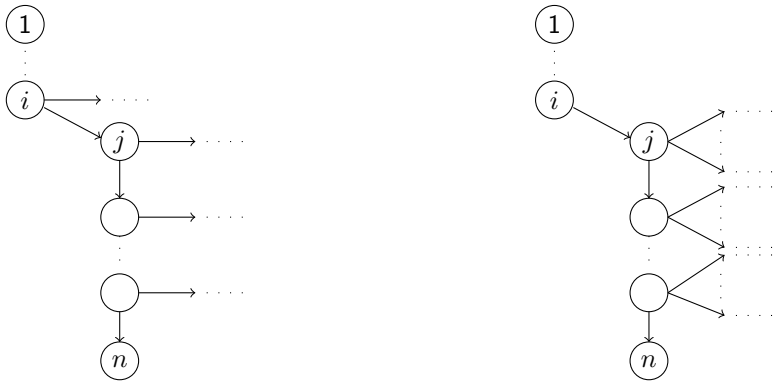
and, thus, the costs in scenario s_r of the paths generated from a $(1, n)$-path p by deviation at node $i \in V(p)$ are sorted, as Figure 2.4.(b) shows, in the following way

$$c_G^{s_r}(q_{i,j_1}^{p,s_r}) \leq \ldots \leq c_G^{s_r}(q_{i,j_{l_i}}^{p,s_r}). \tag{2.6}$$

Assuming that p_{1i} is a simple $(1, i)$-path, the deviation path $q_{i,j}^{p,s_r}$ results from the concatenation of three simple paths and therefore it can still contain repeated nodes. However, the choice of node j can be made in a way that avoids the generation of paths with cycles, by comparing the possible nodes j with the nodes in p_{1i}. Let $i \in V(p)\backslash\{n\}$ and $(i, j) \in A(p)$, the deviation arcs from path p at node i are chosen from the subset of $\mathcal{A}_G^{s_r}(i)$ given by

$$\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j) = \{(i, j_w) \in \mathcal{A}_G^{s_r}(i) : j_w \neq j \,,\, \bar{c}_{ij_w}^{s_r}(G) \geq \bar{c}_{ij}^{s_r}(G) \text{ and } p_{1i} \diamond \langle i, j_w \rangle \text{ is simple } \}.$$

The nodes considered for each path p are those from p's deviation node to the node that precedes n. Figure 2.5.(a). shows a scheme of the deviation paths generated by the MPS algorithm with respect to a path with a deviation arc (i, j).



(a) MPS version (b) Hybrid version

Figure 2.5: Deviation techniques used in the MPS and the hybrid algorithms

In the MPS algorithm, the deviation from a path p at one of its nodes, i, such that $(i, j) \in A(p)$, is obtained by taking the first arc in the ordered set $\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j)$. 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 $\mathcal{A}_G^{s_r}(i)$ is sorted according to non-decreasing order of the reduced costs, for any $i \in V$ [30]. For scenario s_r, the MPS algorithm starts to generate deviations from the shortest path $p^{1,s_r}(G)$ at every of its nodes but n. The resulting paths, one per each scanned node, are stored in a list and are selected, by non-decreasing order of the reduced costs, in future iterations. Each of these paths is identified as the l-th simple shortest path in scenario s_r in case it is simple, for some $l > 1$. This process is repeated under the same conditions. When scanning a path node, at most one new deviation path is generated. The purpose is to avoid the calculation and the storage of unnecessary paths as much as possible, when ranking paths by order of cost. Scanning only the nodes in simple sub-paths reduces the calculation of paths containing cycles, 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 to rank paths, as explained next. With the goal of improving the chances of computing paths with the least robustness cost in an early stage, the new method generates the highest possible number of solution candidates when scanning a path node in the deviation process. An expected consequence is to reduce faster the cost upper-bound and to find an optimal solution quicker than when generating fewer candidates at a time. The deviation technique explored will be similar to the generalization of Yen's algorithm described in [30]. Specifically, the nodes of a path p for scanning are those between its deviation node and node n. Deviating from one of those nodes, i, such that $(i, j) \in A(p)$, consists in generating all deviation paths of form (2.5), with the deviation arcs chosen from set $\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j)$, according to its underlying order. Figure 2.5.(b) illustrates the deviation technique with respect to a path with a deviation arc (i, j).

Since the goal is to determine a robust shortest path that is simple, the generated paths that result from the deviation process must avoid cycles as much as possible. Taking this into account, the deviation process for the hybrid algorithm is performed for the first path $p^{1,s_r}(G)$, by scanning all its nodes but n. For the subsequent deviation paths of form (2.5), all their nodes between their deviation node and node n or the first node which is repeated, are scanned. Every scanned node is the tail node of an arc in $\mathcal{T}_n^{s_r}(G)$. This is valid both for $p^{1,s_r}(G)$, which is a path in that tree, as well as for the paths of the form (2.5), because they result from the concatenation with a path in that tree. Consequently, for a given path p under deviation, any node $i \in V(p)$ that is scanned and $(i, j) \in A(p)$, it holds $\bar{c}_{ij}^{s_r}(G) = 0$. Therefore, (i, j) is the first arc in $\mathcal{A}_G^{s_r}(i)$, i.e. $j = j_1$, and the available deviation arcs with tail node i belong to $\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$. Because this set is sorted, the generated deviation paths q_{i,j_w}^{p,s_r}, $(i, j_w) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$ are ordered

according with (2.6) for the costs in scenario s_r.

In the following, the pruning rules used in Algorithm 1 and the stopping criterion applied for Algorithm 2 are adapted to the hybrid algorithm according to the paths generated at the previous deviation process. Corollary 2.9 rewrites the results in Propositions 2.6 and 2.7 for a generic deviation path, according to the notation introduced in the current section.

Corollary 2.9. *Let $p \in P_{1n}(G)$, $i \in V(p)\backslash\{n\}$, and $q_{i,j}^{p,s_r} = p_{1i} \diamond \langle i,j \rangle \diamond p_{jn}^{1,s_r}(G)$ be the deviation path of p with deviation arc $(i,j) \in A\backslash A(p)$. Let $\tilde{p}$ be any robust shortest path of G containing the sub-path p_{1i}. Then,*

1. $\max_{u \in U_k}\{c_G^{s_u}(p_{1i}) + LB_{in}^{s_u}(G) - LB_{1n}^{s_u}(G)\} \leq RC_G(\tilde{p})$;
2. $c_G^{s_u}(\tilde{p}) \leq c_G^{s_u}(q_{i,j}^{p,s_r})$, *for every* $u \in U_G(q_{i,j}^{p,s_r})$.

The first point of this corollary is a condition for a deviation path to be potentially optimal. 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 correspondent cost of any robust shortest path containing the sub-path p_{1i}. These cost upper-bounds can be combined with the ranking method previously described in order to obtain a robust shortest deviation path. In the following, the results of Corollary 2.9 are applied to establish pruning rules that discard unnecessary deviation paths that do not lead to an optimal solution in G, according with set upper-bounds. These are denoted by UB and $Cmax$ for the robustness cost and for the cost in scenario s_r, respectively, of the paths in $P_{1n}(G)$. Their initial values and the ranking scenario s_r for the deviation paths are determined as in Algorithm 2. According to the deviation process explained above for the hybrid approach, let path $p \in P_{1n}(G)$ be analyzed. Selecting a node $i \in V(p)\backslash\{n\}$, from which the deviation is performed, one has $(i,j_1) \in A(p)$, with (i,j_1) the first arc in $\mathcal{A}_G^{s_r}(i)$. For the generated paths q_{i,j_w}^{p,s_r}, $(i,j_w) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$, the following rules apply:

1. By point 1. of Corollary 2.9, the sub-path p_{1i} does not produce robust shortest deviation paths if
$$\max_{u \in U_k}\{c_G^{s_u}(p_{1i}) + LB_{in}^{s_u}(G) - LB_{1n}^{s_u}(G)\} > UB.$$
In this case, all the deviation paths q_{i,j_w}^{p,s_r}, with $(i,j_w) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$, can be skipped.

2. The previous rule can be refined since, by the same result, if
$$\max_{u \in U_k}\{c_G^{s_u}(p_{1i}) + c_{ij_w}^{s_u}(G) + LB_{j_w n}^{s_u}(G) - LB_{1n}^{s_u}(G)\} > UB,$$
the path $p_{1i} \diamond \langle i,j_w \rangle \diamond p_{j_w n}^{1,s_r}(G)$, $(i,j_w) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$, and its subsequent deviations will not lead to optimal paths, and thus can be skipped.

3. Let $(i, j_{w'})$ be the first arc in $\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$ such that

$$c_G^{s_r}(q_{i,j_{w'}}^{p,s_r}) > Cmax.$$

   Then, all the deviation paths q_{i,j_w}^{p,s_r}, with $(i, j_w) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_{w'}) \cup \{(i, j_{w'})\}$, can be discarded.

4. Let $(i, j_{w''})$ be the first arc in $\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$ such that

$$RC_G(q_{i,j_{w''}}^{p,s_r}) = RD_G^{s_r}(q_{i,j_{w''}}^{p,s_r}) \leq UB.$$

   By point 2. of Corollary 2.9 all paths of form q_{i,j_w}^{p,s_r}, with $(i, j_w) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_{w''})$, and subsequent deviations have a robustness cost not smaller than the optimal value. Therefore, they can be skipped.

The potentially optimal deviation paths are stored in a list X and the path with the least cost in scenario s_r is chosen to be analyzed in the next iteration. Throughout the algorithm, the upper-bounds $Cmax$ and UB are updated. A simple robust shortest path is identified when the stopping criterion used in Algorithm 2 is satisfied.

The main steps of this method are outlined next.

Global algorithmic structure The preliminary procedures for this approach have points in common with both Algorithms 1 and 2. A list W stores the non discarded deviation paths in each iteration and another list X stores all such paths for all iterations. The variables $RCaux$, UB, *sol* and $Cmax$ have the same meaning as in Algorithm 2 and the ranking scenario s_r is determined in the same way. Additionally, the sorted forward star form of the network with respect to the costs in scenario s_r is obtained. The path being scanned is represented by variable p. The deviation node of a new deviation path is represented by i, the arc (i, j_1) corresponds to p's deviation arc, i.e. $p = p_{1i} \diamond \langle i, j_1 \rangle \diamond p_{j_1 n}^{1,s_r}(G)$.

According to the pruning rules described previously, one deviates at a given node i in case 1. or 2. are not satisfied and one may stop deviating at i when a deviation path $q_{i,j}^{p,s_r}$ satisfies 3. or 4., for some $(i, j) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$. Whenever the robustness cost, $RCaux$, of a deviation path improves UB, the latter and the cost upper-bound $Cmax$ are updated. Additionally, one can delete all the paths q in W which exceed the ranking bounds or that would not produce any optimal solution, i.e. that satisfy rules 2. or 3.

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

The pseudo-code of the method described above is presented in Algorithm 3.

Algorithm 3: Hybrid approach for finding a robust shortest path of G

$Q \leftarrow \emptyset$;
for $u \in U_k$ **do**
  Compute $\mathcal{T}_n^{s_u}(G)$; $Q \leftarrow Q \cup \{p^{1,s_u}(G)\}$;
  for $i = 1, \ldots, n-1$ **do** $LB_{in}^{s_u}(G) \leftarrow c_G^{s_u}(p_{in}^{1,s_u}(G))$;
$UB \leftarrow \min\{RC_G(q) : q \in Q\}$;
$sol \leftarrow q$ such that $q \in Q$ and $RC_G(q) = UB$;
$r \leftarrow \min\left\{u \in U_k : RD_G^{s_u}(sol) = UB\right\}$;
$Cmax \leftarrow c_G^{s_r}(sol)$; $X \leftarrow \emptyset$; $p \leftarrow p^{1,s_r}(G)$;
Store the network in the sorted forward star form with respect to the costs for scenario s_r;
while *there exists a path p to be scanned such that* $RD_G^{s_r}(p) \neq UB$ **do**
  $W \leftarrow \emptyset$;
  for $i \in V(p)$ *from the head node of p's deviation arc to the node that precedes n* **do**
    $p_{1i} \leftarrow (1,i)$-sub-path of p;
    if p_{1i} *is not simple* **then break**;
    if $\max_{u \in U_k}\{c_G^{s_u}(p_{1i}) + LB_{in}^{s_u}(G) - LB_{1n}^{s_u}(G)\} > UB$ **then break**;
    $j_1 \leftarrow$ head node of p's arc with tail node i;
    $\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1) \leftarrow \{(i, j_w) \in \mathcal{A}_G^{s_r}(i) : w > 1$ and $p_{1i} \diamond \langle i, j_w\rangle$ is simple$\}$;
    for $(i,j) \in \hat{\mathcal{A}}_G^{s_r}(p_{1i}, j_1)$ **do**
      if $\max_{u \in U_k}\{c_G^{s_u}(p_{1i}) + c_{ij}^{s_u}(G) + LB_{jn}^{s_u}(G) - LB_{1n}^{s_u}(G)\} \leq UB$ **then**
        $q_{i,j}^{p,s_r} \leftarrow p_{1i} \diamond \langle i,j\rangle \diamond p_{jn}^{1,s_r}(G)$;
        if $c_G^{s_r}(q_{i,j}^{p,s_r}) > Cmax$ **then break**;
        $W \leftarrow W \cup \{q_{i,j}^{p,s_r}\}$;
        $RCaux \leftarrow \max\{c_G^{s_u}(p_{1i}) + c_{ij}^{s_u}(G) + LB_{jn}^{s_u}(G) - LB_{1n}^{s_u}(G) : u \in U_k\}$;
        if $RCaux < UB$ **then**
          $UB \leftarrow RCaux$; $Cmax \leftarrow LB_{1n}^{s_r}(G) + UB$;
          Delete from W any path q such that $c_G^{s_r}(q) > Cmax$, or $\max_{u \in U_k}\{c_G^{s_u}(q_{1i}) + c_{ij}^{s_u}(G) + LB_{jn}^{s_u}(G) - LB_{1n}^{s_u}(G)\} > UB$, such that (i,j) is q's deviation arc;
        if $RD_G^{s_r}(q_{i,j}^{p,s_r}) = UB$ **then break**;
  if $RC_G(p) = UB$ **and** p is simple **then** $sol \leftarrow p$;
  Delete from X any path q such that $c_G^{s_r}(q) > Cmax$, or $\max_{u \in U_k}\{c_G^{s_u}(q_{1i}) + c_{ij}^{s_u}(G) + LB_{jn}^{s_u}(G) - LB_{1n}^{s_u}(G)\} > UB$, such that (i,j) is q's deviation arc;
  $X \leftarrow X \cup W$;
  $p \leftarrow$ shortest path for scenario s_r in X; $X \leftarrow X - \{p\}$;
return sol;

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 $O_1^a = \mathcal{O}(km + k^2n)$ for acyclic networks and $O_1^c = \mathcal{O}(k(m + n\log n) + k^2n)$ for general networks. Before the ranking starts, the arc costs are replaced by their reduced costs, $\mathcal{O}(m)$, and the network is represented in the sorted forward star form, $\mathcal{O}(m\log n)$ [30].

The second phase concerns the deviation process. Assume H paths are ranked, that is, the **while** loop in line 10 is performed H times. In the worst case, scanning one path p demands scanning all the arcs of G, trying to generate new deviations. The costs of all father sub-paths in p to be deviated for all scenarios are obtained in $\mathcal{O}(kn)$ time. When deviating from each node $i \in V(p)$, the first pruning rule is tested once in $\mathcal{O}(k)$ time. The second rule involves the analysis of the extension of each path $p_{1i} \diamond \langle i, j\rangle$, with (i, j) the deviation arc, by testing the condition at line 19 in $\mathcal{O}(k)$ time. The third rule involves the calculation of $c_G^{s_r}(q_{i,j}^{p,s_r})$ and its comparison with $Cmax$ in $\mathcal{O}(1)$. The fourth rule implies the calculation of $RC_G(q_{i,j}^{p,s_r})$, $\mathcal{O}(k)$, and its comparison with UB, in $\mathcal{O}(1)$. Hence, the total amount of work for each path is $\mathcal{O}(km + kn) = \mathcal{O}(km)$, and the second phase has a complexity of $\mathcal{O}(Hkm)$.

In these circumstances, the total complexity is $O_1^a + O_2^a = \mathcal{O}(k^2n + kmH + m\log n)$ for acyclic networks, and $O_1^c + O_2^c = \mathcal{O}(k^2n + k\max\{mH, n\log n\} + m\log n)$ for general networks. Like parameter L used in Algorithm 2, H depends on each problem's dimension and cannot be known in advance.

Example Like for Algorithms 1 and 2, network G_3 of Figure 2.1 is considered for illustrating the application of Algorithm 3. Let be recalled that Figure 2.2 represents the trees $\mathcal{T}_6^u(G_3)$, $u \in U_2$. The hybrid method has the same initialization procedure of Algorithm 2 before ranking the paths. Analogously,

$$UB = 12 \ ; \ sol = \langle 1, 3, 6\rangle \ \text{ and } \ Cmax = 52$$

are set and scenario 1 is chosen to rank the deviation paths.

Figure 2.6 shows the tree of the paths obtained by Algorithm 3. The first path under deviation is

$$p = p^{1,1}(G_3) = \langle 1, 2, 4, 6\rangle,$$

by scanning all its nodes but 6. Taking into account that

$$\hat{\mathcal{A}}_{G_3}^1(\langle 1\rangle, 2) = \{(1, 3)\} \ ; \ \hat{\mathcal{A}}_{G_3}^1(\langle 1, 2\rangle, 4) = \emptyset \ \text{ and } \ \hat{\mathcal{A}}_{G_3}^1(\langle 1, 2, 4\rangle, 6) = \emptyset,$$

the only path outputted on the first iteration of the method is $q_{1,3}^{p,1} = \langle 1, 3, 1, 2, 4, 6\rangle$, with

$$c_{G_3}^1(q_{1,3}^{p,1}) = 40 < Cmax \ \text{ and } \ RC_{G_3}(q_{1,3}^{p,1}) = 26 \geq UB.$$

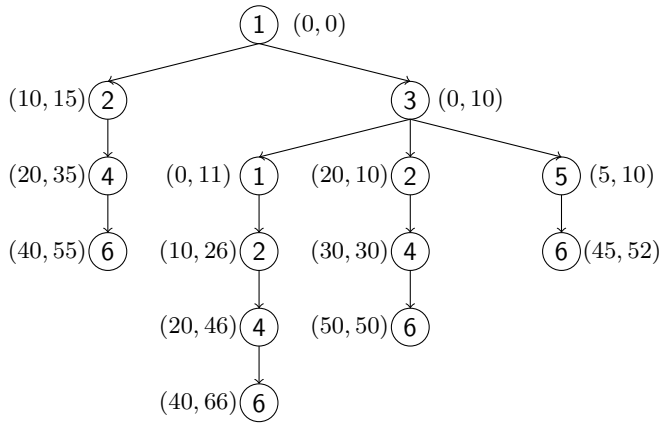


Figure 2.6: Tree of the deviation paths produced by Algorithm 3 for G_3

Hence, in the following iteration, the path

$$p = q_{1,3}^{p,1} = \langle 1, 3, 1, 2, 4, 6 \rangle$$

is considered. Because p contains a cycle, only node 3 is scanned. In fact, potentially optimal paths can be produced with such deviation, given that

$$\max_{u \in U_2} \left\{ c_G^u(\langle 1, 3 \rangle) + LB_{36}^u(G_3) - LB_{16}^u(G_3) \right\} = 0 \leq UB.$$

The set of possible deviation arcs is given by

$$\hat{\mathcal{A}}_{G_3}^1(\langle 1, 3 \rangle, 1) = \{(3, 5), (3, 2), (3, 6)\}.$$

When using deviation arc $(3, 5)$, the simple path $q_{3,5}^{p,1} = \langle 1, 3, 5, 6 \rangle$ is computed, with

$$c_{G_3}^1(q_{3,5}^{p,1}) = 45 < Cmax \quad \text{and} \quad RC_{G_3}(q_{3,5}^{p,1}) = 12,$$

where the last equality does not improve UB. Path $\langle 1, 3, 5, 6 \rangle$ is stored in list W and the deviation path $q_{3,2}^{p,1} = \langle 1, 3, 2, 4, 6 \rangle$ is obtained. This path satisfies

$$c_{G_3}^1(q_{3,2}^{p,1}) = 50 < Cmax \quad \text{and} \quad RC_{G_3}(q_{3,2}^{p,1}) = 10 < UB,$$

so it is a new candidate for optimality, which is stored in W. Then, the variables UB and $Cmax$ are updated to

$$UB = 10 \quad \text{and} \quad Cmax = 50.$$

Moreover, since

$$RD_{G_3}^1(q_{3,2}^{p,1}) = UB,$$

future deviations from path p at node 3 do not improve the best robustness cost obtained so far, and, therefore, the deviation arc $(3, 6)$ is not considered. In spite of

$$c^1_{G_3}(\langle 1, 3, 5, 6\rangle) = 45 < Cmax,$$

path $\langle 1, 3, 5, 6\rangle$ satisfies

$$\max_{u \in U_2}\{c^u_{G_3}(\langle 1, 3\rangle) + c^u_{35}(G_3) + LB^u_{56}(G_3) - LB^u_{16}(G_3)\} = 11 > UB.$$

Thus, this path will not produce optimal deviation paths and it is removed from W. Then, the only path left in W, $\langle 1, 3, 2, 4, 6\rangle$, is transferred to the empty list X. Because

$$\hat{\mathcal{A}}^1_{G_3}(\langle 1, 3, 2\rangle, 4) = \hat{\mathcal{A}}^1_{G_3}(\langle 1, 3, 2, 4\rangle, 6) = \emptyset,$$

no new deviation arcs exist. Therefore, $\langle 1, 3, 2, 4, 6\rangle$ updates sol, because it is simple and its robustness cost is UB, being returned as the robust shortest path of G_3.

It is worth noting that, compared with the simple paths ranked by Algorithm 2, the computation of $\langle 1, 3, 5, 4, 6\rangle$ is skipped in Algorithm 3.

2.5 Computational experiments

This section is devoted to the empirical evaluation of the methods presented in the preceding sections and to their comparison with the exact algorithm by Yu and Yang [50]. This latter approach is based on dynamic programming. It applies a recursive relation to find the optimal value of the problem, starting with a cost upper-bound for each scenario, given by the sum of the costs of the $n - 1$ arcs with the largest costs. This makes the method particularly sensitive to the range of cost values and to the variation of the number of scenarios.

From now on, Algorithms 1, 2 and 3 will be represented by LA, RA and HA, respectively, whereas YA will represent Yu and Yang's algorithm. In order to evaluate and to compare the performances of these algorithms, they were implemented in Matlab 7.12. and ran on a computer equipped with an Intel Pentium Dual CPU T2310 1.46GHz processor and 2GB of RAM. The implementations of LA, RA and HA use Dijkstra's algorithm [1] to compute the trees $\mathcal{T}^{s_u}_n(G)$, $u \in U_k$. In LA, X is managed as a FIFO list. The MPS algorithm [30] is applied to rank the simple paths in RA.

The benchmarks used in the experiments correspond to randomly generated directed networks. For each network dimension, 10 problems were generated and solved by each of the aforementioned algorithms. The average and the standard deviation of the total CPU times (registered in seconds) are denoted by Ave_t and Std_t, respectively.

A first set of tests intended to compare the new methods with *YA*. These tests ran for random networks with $n \in \{5, 10\}$, $d \in \{2, 3, n-1\}$, $k \in \{2, 3\}$ and costs generated in $U(0, 20)$. The obtained total CPU times are summarized on Table 2.2.

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

Table 2.2: Total CPU times (in seconds)

For these cases, *YA* has a poor performance when compared with the other three methods, reporting results 10^4 or 10^5 times bigger than the corresponding ones for *LA*, *RA* and *HA*, which had all a similar behavior. Such difference increases with the number of scenarios. For $k \geq 4$, the tests for *YA* ran too slowly, therefore the results are omitted.

Because of the previous results with YA, the second set of tests only comprised the codes *LA*, *RA* and *HA*. The considered instances were bigger than the previous set and the costs were generated in $U(0, 100)$. For $k \in \{2, 3, 4, 5, 10, 50, 100, 500, 1000, 5000\}$, the computational experiments were performed over complete networks, with $n \in \{5, 10, 15\}$, and over random networks, with $n \in \{250, 500, 750\}$ and $d \in \{5, 10, 15\}$. For these tests, the total CPU time is split into the time needed to compute the trees $\mathcal{T}_n^{s_u}(G)$, $u \in U_k$, and the time required for the remaining procedures. The averages of the partial times are denoted by Ave_{t_i}, $i = 1, 2$, and they are reported in Table 2.3 for complete networks and in Table 2.4 for random networks. Some of the values are omitted, whenever the codes were too slow. In terms of the total CPU time, $Ave_t = Ave_{t_1} + Ave_{t_2}$.

Let Ave_r and Ave_h represent the average number of simple $(1, n)$-paths ranked in *RA* and *HA*, respectively, and Std_r and Std_h be the corresponding standard deviations. Such results are presented in Table 2.5, for complete networks, and in Table 2.6, for random networks. The averages of the total number of computed simple paths are then $Ave_{p_r} = Ave_r + k$ and $Ave_{p_h} = Ave_h + k$. Analogously, the associate standard deviations are $Std_{p_r} = Std_r$ and $Std_{p_h} = Std_h$.

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

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

Table 2.3: Partial CPU times for complete networks (in seconds)

			LA	RA	HA		LA	RA	HA		LA	RA	HA
		$n = 250$				$n = 500$				$n = 750$			
d	k	$Ave_{t_1}^{(*)}$	Ave_{t_2}	Ave_{t_2}	Ave_{t_2}	$Ave_{t_1}^{(*)}$	Ave_{t_2}	Ave_{t_2}	Ave_{t_2}	$Ave_{t_1}^{(*)}$	Ave_{t_2}	Ave_{t_2}	Ave_{t_2}
5	2	0.243	0.172	0.061	0.055	0.486	0.368	0.152	0.148	0.967	1.582	0.380	0.329
	3	0.374	0.274	0.117	0.057	0.718	0.322	0.401	0.166	1.393	1.051	0.537	0.348
	4	0.487	0.253	0.407	0.069	1.022	0.506	0.767	0.180	2.352	1.117	1.014	0.430
	5	0.580	0.205	1.085	0.077	1.316	0.433	1.179	0.208	2.492	1.346	2.015	0.443
	10	1.152	0.290	5.024	0.117	2.276	0.340	14.091	0.268	4.914	1.402	32.518	0.634
	50	6.239	0.473	166.560	0.528	11.990	0.876	393.408	1.013	23.491	1.562	588.175	1.629
	100	11.052	0.899	243.745	1.074	23.050	1.607	508.091	2.228	50.008	2.768	1695.842	3.727
	500	51.618	7.662	1332.511	9.734	108.264	10.247	5177.183	13.169	233.830	14.968	6256.625	19.879
	1000	102.774	24.211	1078.863	28.347	301.306	33.695	3999.946	39.847	397.955	40.415	7532.278	51.006
	5000	514.241	522.163	3480.725	568.822	1190.716	631.577	10060.447	719.241	1873.489	627.860	–	712.522
10	2	0.268	0.479	0.259	0.057	0.599	0.764	0.428	0.170	0.822	1.740	0.597	0.321
	3	0.371	0.350	17.300	0.074	0.855	0.720	31.129	0.198	1.491	2.357	46.015	0.390
	4	0.489	0.376	204.664	0.084	1.084	0.673	491.860	0.236	1.978	2.496	312.169	0.433
	5	0.587	0.408	531.604	0.089	1.449	1.063	3625.298	0.282	2.434	3.189	3087.081	0.526
	10	1.242	0.147	7694.775	0.146	2.898	0.639	–	0.381	4.239	2.195	–	0.659
	50	5.883	0.554	–	0.712	13.586	1.288	–	1.523	21.585	3.553	–	3.617
	100	11.238	1.060	–	1.633	28.917	2.718	–	3.778	41.405	4.681	–	6.179
	500	56.614	8.883	–	11.199	130.719	13.519	–	18.441	215.853	17.708	–	26.255
	1000	120.067	28.493	–	33.852	316.801	38.556	–	49.188	462.043	52.427	–	73.979
	5000	577.927	655.516	–	795.650	1346.546	707.249	–	743.674	2041.511	824.071	–	977.557
15	2	0.328	0.453	3.284	0.074	0.586	1.461	2.769	0.175	0.900	2.665	12.107	0.351
	3	0.407	0.591	598.616	0.090	0.909	2.028	2209.236	0.245	1.422	3.477	2195.500	0.385
	4	0.547	0.639	11256.692	0.112	1.151	1.842	–	0.343	1.740	3.385	–	0.461
	5	0.668	0.420	–	0.111	1.447	1.223	–	0.337	2.140	3.990	–	0.502
	10	1.273	0.548	–	0.190	2.839	1.572	–	0.581	4.447	3.356	–	0.752
	50	6.066	0.710	–	0.673	15.486	1.073	–	1.458	26.819	1.838	–	2.150
	100	13.021	1.000	–	1.381	36.128	2.308	–	2.981	45.036	2.670	–	4.088
	500	69.681	10.187	–	12.077	165.870	14.892	–	20.634	264.081	16.705	–	23.343
	1000	135.152	29.934	–	35.643	332.505	42.902	–	53.244	497.384	44.877	–	65.762
	5000	599.005	698.254	–	727.915	1325.300	746.045	–	783.147	2226.412	1093.321	–	1343.007

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

Table 2.4: Partial CPU times for random networks (in seconds)

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

Table 2.5: Number of ranked simple paths for complete networks

According to Tables 2.3 and 2.4, computing the trees $\mathcal{T}_n^{s_u}(G)$, $u \in U_k$, 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 ($k \geq 500$) or random networks with $n = 250$ and $k = 5000$. Nevertheless, LA presented other exceptions for random networks with few scenarios ($k \leq 5$).

In general, most of RA's time was invested on the second step, namely when the number of ranked paths or the number of path deviation costs demanded a major computational effort. The latter cases are reflected in the results obtained for all types of networks when $k \geq 100$ and the former stand for the denser networks, like complete networks with $n \in \{10, 15\}$ and random networks with $d = 15$. In fact, the higher the density of a network, the more arcs emerge from each node, which increases the chances of computing a large number of simple paths till a solution is obtained, as Tables 2.5 and 2.6 show. Moreover, since Ave_r was always greater than Ave_h, $Ave_{t_2}(RA)$ was also always greater than $Ave_{t_2}(HA)$, even for the cases where the second phase had a minor role in the performance of RA. This was the case for complete networks with $n = 5$ and $k < 100$, where in average up to 3 simple paths were ranked, and for random networks with small densities and few scenarios, as $d = 5$ and $k \in \{2, 3, 4\}$ or $d = 10$ and $k = 2$.

Tables 2.7, 2.8 and 2.9 show the averages and the standard deviations of the total CPU times for complete and for random networks, respectively. Like before, large CPU times are omitted.

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

Table 2.6: Number of ranked simple paths for random networks

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

Table 2.7: Total CPU times for complete networks (in seconds)

			LA		*RA*		*HA*	
n	d	k	Ave_t	Std_t	Ave_t	Std_t	Ave_t	Std_t
250	5	2	0.415	0.051	0.304	0.041	0.298	0.031
		3	0.648	0.208	0.491	0.061	0.431	0.106
		4	0.740	0.083	0.894	0.669	0.556	0.023
		5	0.785	0.078	1.665	1.345	0.657	0.018
		10	1.442	0.179	6.176	6.459	1.269	0.082
		50	6.712	0.558	172.799	305.654	6.767	0.974
		100	11.951	0.572	254.797	325.766	12.126	0.703
		500	59.280	2.572	1384.129	2837.640	61.352	2.140
		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 2.8: Total CPU times for random networks with $n \in \{250, 500\}$ (in seconds)

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

Table 2.9: Total CPU times for random networks with $n = 750$ (in seconds)

Code *HA* outperformed *RA* for all cases in terms of time, as shown in Tables 2.7, 2.8 and 2.9. Nevertheless, *HA* was not always the most efficient method, given that *LA* had the best performance in problems with $k \geq 100$. The standard deviations of the total CPU times provide information about the variability on the results towards the averages. In this sense, *LA* and *HA* were the most stable codes, with standard deviations generally smaller than the corresponding averages. Instead, *RA* had the most irregular performance due to the high values of $Std_t(RA)$, usually greater than $Ave_t(RA)$. This is supported by the high variability of the number of paths ranked by *RA* on Tables 2.5 and 2.6. In contrast, the number of paths ranked by *HA* did not vary much and the averages Ave_h are quite small, especially when $k \geq 50$ for denser networks, where only one iteration was needed to obtain the optimal solution.

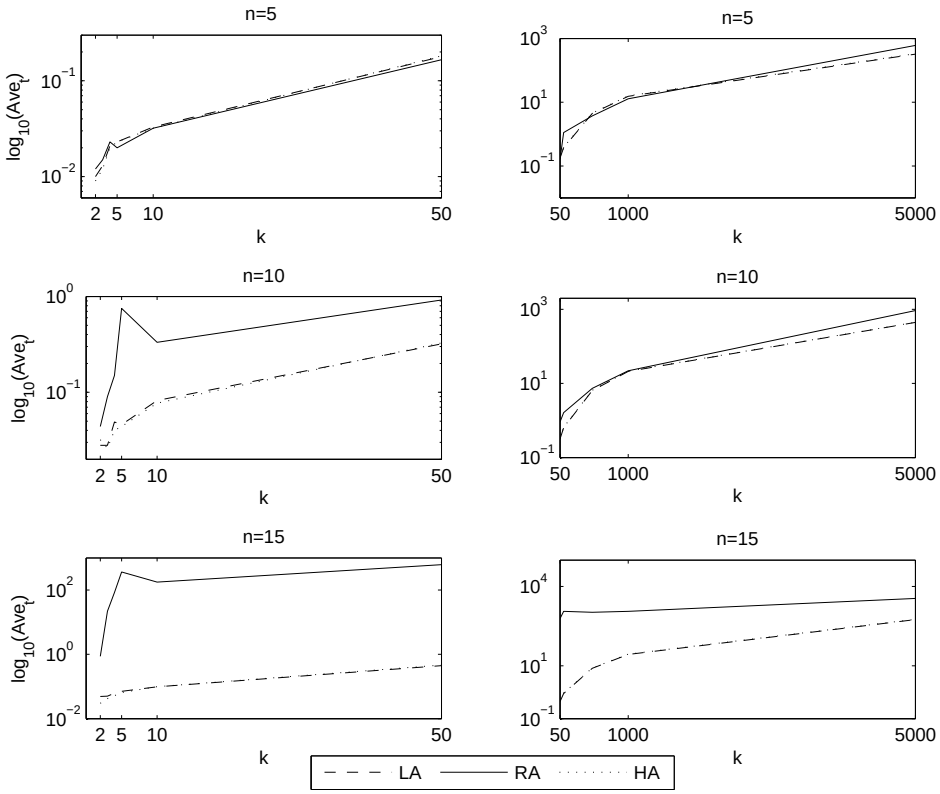


Figure 2.7: Total CPU times for complete networks

The behavior of the average CPU times can be evaluated by varying a single parameter at a time. When n and d are fixed, Tables 2.3 and 2.4 show that the average time for computing the trees $\mathcal{T}_n^{s_u}(G)$, $u \in U_k$, grows when k increases, which is explained by the increase of the number of shortest paths for the scenarios of G ending at node n. In what concerns the second phase of the algorithms, $Ave_{t_2}(LA)$ and $Ave_{t_2}(HA)$ showed the smoothest growths. Instead, $Ave_{t_2}(RA)$

increased more irregularly with k, due to the unstable variation of Ave_r in Tables 2.5 and 2.6. As computing the trees $\mathcal{T}_n^{s_u}(G)$, $u \in U_k$, is the common initial task for all algorithms, their behavior on the second phase mimics the evolution of their average total CPU times. The plots in Figures 2.7 and 2.8 show these 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 $n = 750$ for RA.

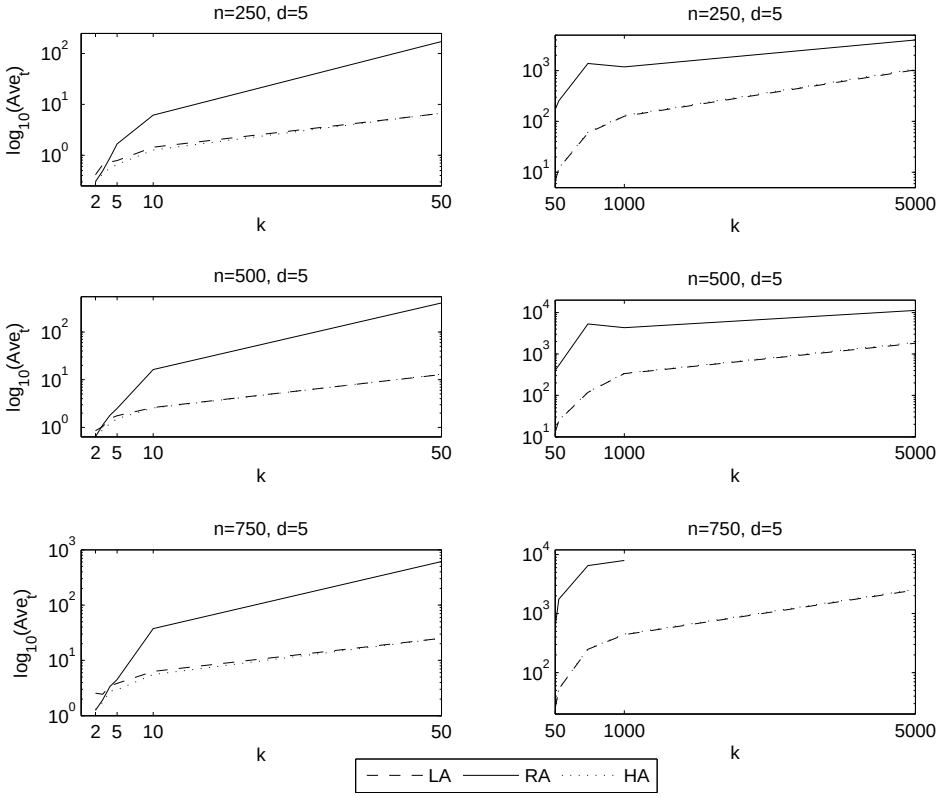


Figure 2.8: Total CPU times for random networks with $d = 5$

The averages $Ave_t(LA)$ and $Ave_t(HA)$ grew similarly with the increase of k but slower than $Ave_t(RA)$. The increase of the latter is steeper than for LA and HA and it is quite irregular for small values of k, due to the unsteady behavior of the ranking. Moreover, all averages increase slower when $k > 1000$, since all the performances become more dependent on the cost calculations.

The obtained results may also be analyzed from the perspective of fixing the number of scenarios. Based on Figures 2.7 and 2.8, for different values of k, $Ave_t(LA)$ and $Ave_t(HA)$ become more distant from $Ave_t(RA)$ when n increases. This results from the growth of the

number of paths in G with n, which may also affect 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 2.8 and 2.9, as well as by the plots in Figure 2.9 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 $n = 250$.

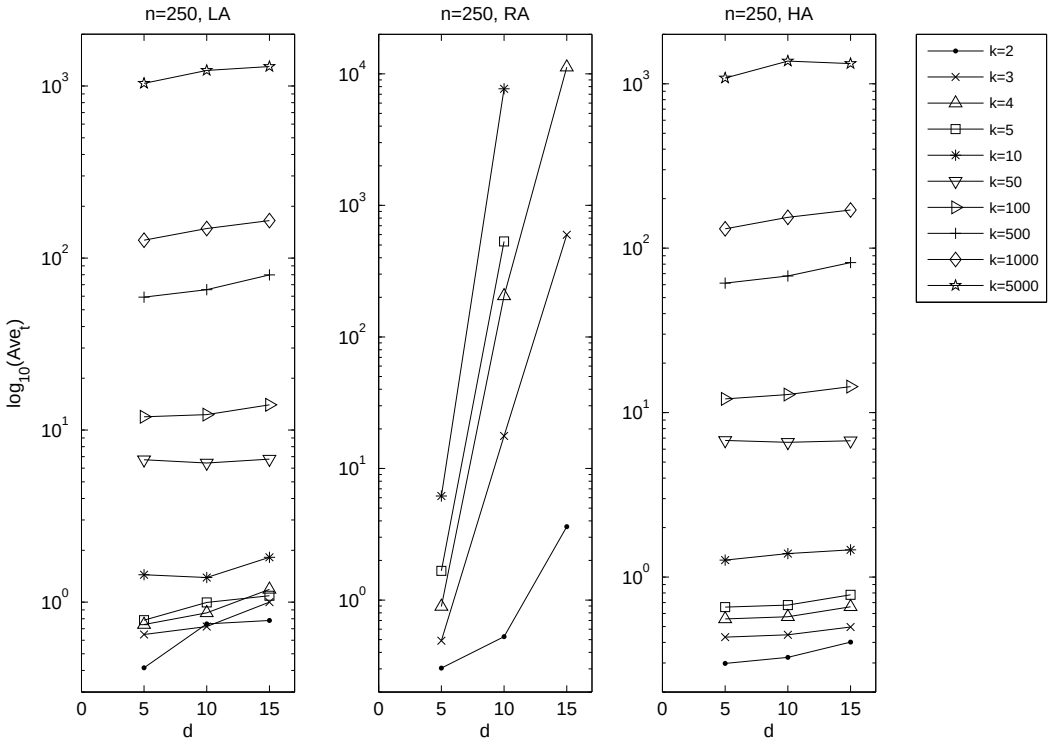


Figure 2.9: Total CPU times for random networks with $n = 250$

In general, LA and HA were the most effective methods to solve the robust shortest path problem, showing similar behaviors. The experiments showed that HA was the fastest when the number of scenarios did not exceed 50, solving the problem in less than half minute in average, whereas LA was the best alternative for networks with 1000 or 5000 scenarios, running in less than one hour in average.

2.6 Conclusions

Three algorithms were presented to solve the robust shortest path problem with a finite number of scenarios. These algorithms were introduced in [39]. All of them allow to obtain a simple optimal solution and improve the approach of Yu and Yang [50], the only exact method known. The first is a labeling approach, the second is based on the ranking of simple paths and the third is a hybrid version of the previous, which combines pruning techniques from both while ranking simple paths in a specific manner. The novelty of the hybrid algorithm when compared to a simple version of the ranking based method is twofold. On the one hand, its pruning rules allow to skip uninteresting paths; while, on the other, it promotes the early generation of more candidate paths than with a standard implementation, seeking to produce good cost upper-bounds. This strengthens the elimination of bad solutions.

The developed methods have time complexity orders depending on the network parameters, as well as on the maximum number of generated paths in each set $P_{1i}(G)$, $i \in V$, in the first method, and on the number of ranked paths in the second and the third.

Implementations of the three methods were tested on randomly generated networks and compared with the algorithm by Yu and Yang [50]. The new approaches were more efficient than the latter, and revealed to be effective over problems handling with large cost upper-bounds or a large number of scenarios. In spite of this progress, out of the three introduced methods, the ranking approach was the one with the poorest performance, due to the variable number of simple paths that had to be ranked before the robust shortest path could be obtained. The changes introduced in the hybrid version resulted in an improvement of this step and, thus, of the initial version of the algorithm. The labeling and the hybrid methods had similar behaviors and showed to be quite effective for solving the robust shortest path problem with up to 1000 scenarios. Nevertheless, the labeling algorithm stood out for problems with 1000 or 5000 scenarios, running in less than one hour in average, while the hybrid algorithm was the best when the number of scenarios did not exceed 50, solving the problem in less than half minute in average.

Chapter 3

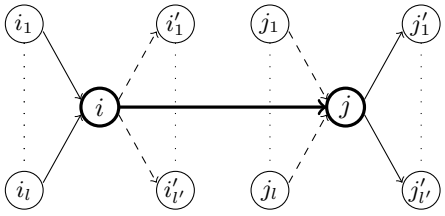
Preprocessing techniques for the robust shortest path problem

The present chapter is dedicated to the development of methods that allow to simplify a given robust shortest path problem before it is solved, and thus making easier the search for an optimal solution. This is achieved by means of techniques, named preprocessing techniques, aimed at reducing the network in such a way that the result still contains all the optimal solutions in the original network. The preprocessing techniques developed here focus two aspects: the identification of nodes (or arcs) that belong to all optimal solutions, and the identification of nodes (or arcs) that do not belong to any optimal solution. The application of such techniques is based on the comparison of particular path robust deviations in specific scenarios and an established lower-bound. Two versions of the preprocessing rules are developed, a static version and a dynamic version. In the former, the cost lower-bound is set at the beginning of the method, and it remains unchanged while network nodes/arcs are scanned, whereas in the latter that value is updated as the scan is performed. The cases of nodes and arcs are studied separately. First, the theoretical results are presented, then the pseudo-codes of the algorithms are outlined and their complexity order in terms of the number of operations is calculated. Afterwards, the new methods are exemplified for small networks, and the results of computational experiments are reported and discussed.

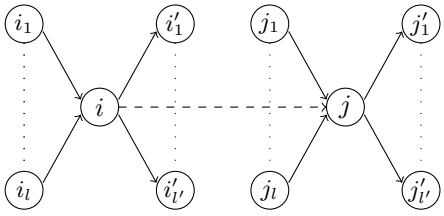
3.1 Introduction

As mentioned above, the main purpose of this chapter is to identify parts of a network that can be deleted, ensuring that the resulting network still contains all the robust shortest paths of the former. With this goal in mind, a node (arc) is called robust 1-persistent if it certainly belongs to any optimal solution, and it is called robust 0-persistent if it does not belong to any

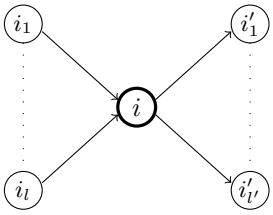
of them.



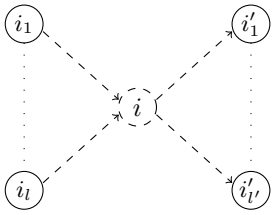
(a) arc (i, j) is robust 1-persistent



(b) arc (i, j) is robust 0-persistent



(c) node i is robust 1-persistent



(d) node i is robust 0-persistent

Figure 3.1: Identification of arcs/nodes for deletion under robust 1 or 0- persistency

The two types of persistency, for nodes or for arcs, lead to different conclusions, summarized in Figure 3.1. Because robust 1-persistent nodes (arcs) necessarily belong to any robust shortest path, they cannot be deleted from the network. Furthermore, knowing in advance that a node is robust 1-persistent does not make the problem easier – see Figure 3.1.(c). On the contrary, if the arc (i, j) is robust 1-persistent, no other arcs starting in i or ending in j belong to an optimal solution, which is simple, and therefore they can be deleted. This is shown in Figure 3.1.(a), where the dashed lines represent arcs/nodes that can be deleted and the solid thick lines are arcs/nodes in the optimal solutions.

Additionally, robust 0-persistent nodes (arcs) do not belong to any optimal solution and, thus, they can be deleted from the network. For any of these nodes, i, this result can be extended to all the arcs that emerge or end at i, as shown in Figure 3.1.(d). However, no further conclusions can be drawn when a robust 0-persistent arc is found – Figure 3.1.(b).

Based on the remarks above, finding robust 1-persistent arcs (0-persistent nodes) is expected to be more effective, in terms of the network reduction, than identifying robust 1-persistent nodes (0-persistent arcs). For this reason, the rest of the chapter will focus only on the identification of robust 1-persistent arcs and robust 0-persistent nodes.

Preprocessing techniques have been addressed for the robust shortest path problem for

continuous models with interval data. The first of these rules were introduced by Karasan et al. [26], for detecting and eliminating robust 0-persistent arcs in acyclic networks. One decade later, Catanzaro et al. [13] extended the reduction of the interval robust shortest path problem to general networks, by detecting robust 0-persistent arcs and nodes. Moreover, these authors combined the latter strategy with the identification of robust 1-persistent arcs. The combination of the two techniques significantly reduced the size of the problem on a set of computational experiments.

The results presented in this chapter are inspired in [13]. They aim at identifying robust 1-persistent arcs and robust 0-persistent nodes for the robust shortest path problem in the case of a finite set of scenarios. First, conditions that all robust 1-persistent arcs and robust 0-persistent nodes should satisfy are established. We propose two different algorithmic approaches for applying these tests. The first approach, introduced in [40], consists of static algorithms that check the above conditions for the arcs and nodes in the network. The conditions are established at the beginning of the algorithm and are based on a lower-bound for the robustness cost obtained from the shortest paths in all scenarios. The second approach, introduced in [41], uses a dynamic search, with the goal of identifying more robust 1-persistent arcs (or robust 0-persistent nodes). The same conditions used in the static version are tested, however, the parameter they depend on is updated as paths are listed. The number of scenarios considered in the comparisons when preprocessing nodes is also taken into consideration, in order to reduce the number of tests to perform.

3.2 Identification of robust 1-persistent arcs

As said before, the identification of robust 1-persistent arcs was treated in [13], in the context of interval data problems. This strategy was not particularly effective, according with the computational tests. However, together with the detection of robust 0-persistent arcs or nodes, it led to a significant reduction of the network. Under these conditions, an optimal solution could be found more easily than without preprocessing robust 1-persistent arcs. For the new result, these arcs were chosen among the arcs of the shortest $(1, n)$-path in the scenario associated with the upper-limits of the intervals. In addition, the scenario that assigned the upper-limits of the interval costs to the arcs of that path and the lower-limits of the interval costs to the remaining arcs of the network was also considered. Then, the rule was derived by determining the shortest $(1, n)$-path in the subnetwork resultant from removing the arc under analysis at the original network, in case node n was still reachable from node 1.

The following result has a similar motivation and introduces a broader rule for detecting robust 1-persistent arcs, which are restricted to the shortest $(1, n)$-paths for the scenarios of G.

Provided that a $(1,n)$-path and its robustness cost are known, it deals with the scenarios for which the associate shortest $(1,n)$-paths of G contain the arc (i,j) under evaluation and with the shortest $(1,n)$-path in the subnetwork of G resultant from the removal of (i,j), $G^-_{(i,j)}$.

Proposition 3.1. *Let $q \in P_{1n}(G)$ be a path such that $A(q) \cap A(p^{1,s_u}(G)) \neq \emptyset$, for some $u \in U_k$. Let $(i,j) \in A(q) \cap \left(\bigcup_{u=1}^{k} A(p^{1,s_u}(G)) \right)$ be an arc, such that node n is reachable from node 1 in $G^-_{(i,j)}$, and $U(i,j) = \{u \in U_k : (i,j) \in p^{1,s_u}(G)\}$ be the set of scenarios for which the shortest $(1,n)$-paths of G contain arc (i,j). If*

$$\exists u' \in U(i,j) : \; RD_G^{s_{u'}}(p^{1,s_{u'}}(G^-_{(i,j)})) > RC_G(q),$$

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

Proof. Let $q \in P_{1n}(G)$, $(i,j) \in A(q) \cap \left(\bigcup_{u=1}^{k} A(p^{1,s_u}(G)) \right)$ and $p \in P_{1n}(G^-_{(i,j)})$. By definition of robustness cost of a $(1,n)$-path and because $p^{1,s_u}(G^-_{(i,j)})$ is the shortest $(1,n)$-path in G that does not contain arc (i,j), under scenario s_u, $u \in U_k$,

$$RC_G(p) = \max_{u \in U_k} RD_G^{s_u}(p) \geq \max_{u \in U_k} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})), \tag{3.1}$$

with

$$\max_{u \in U_k} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})) = \max \left\{ \max_{u \in U_k \backslash U(i,j)} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})), \max_{u \in U(i,j)} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})) \right\}.$$

For every $u \in U_k \backslash U(i,j)$, one has $p^{1,s_u}(G^-_{(i,j)}) = p^{1,s_u}(G)$, and, therefore,

$$RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})) = RD_G^{s_u}(p^{1,s_u}(G)) = 0.$$

This means that $\max_{u \in U_k \backslash U(i,j)} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})) = 0$. Since any robust deviation of a $(1,n)$-path is non-negative, one has

$$\max_{u \in U_k} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})) = \max_{u \in U(i,j)} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})).$$

Then, (3.1) implies

$$RC_G(p) \geq \max_{u \in U(i,j)} RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})).$$

Hence, if $RD_G^{s_{u'}}(p^{1,s_{u'}}(G^-_{(i,j)})) > RC_G(q)$, for some $u' \in U(i,j)$, by hypothesis,

$$RC_G(p) > RC_G(q).$$

This means that any path in $P_{1n}(G)$ that does not contain arc (i,j) cannot be a robust shortest path. Therefore, arc (i,j) is robust 1-persistent. □

The condition of Proposition 3.1 can be simplified in terms of notation. Let $RCmin$ be a variable which sets its lower-bound, Arc denote the set of arcs to be scanned and the path robust deviations be represented as

$$RDA^{u}_{(i,j)} = RD^{s_u}_{G}(p^{1,s_u}(G^{-}_{(i,j)}))\,,\ (i,j) \in A\,,\ u \in U(i,j),\ \text{such that } p^{1,s_u}(G^{-}_{(i,j)}) \text{ exists.}$$

Then, Proposition 3.1 can be rewritten, considering that, for any arc $(i,j) \in Arc$, if

$$\exists u' \in U(i,j):\ RDA^{u'}_{(i,j)} > RCmin, \tag{3.2}$$

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

In the following, the static and dynamic algorithms to identify robust 1-persistent arcs will be introduced, starting with their common procedures.

The variable $RCmin$ is initialized with the minimum robustness cost of the shortest $(1,n)$-paths for the scenarios of G, analogously to what was considered for the algorithms in Chapter 2. For that, list Q is used to store only the distinct shortest $(1,n)$-paths and

$$RCmin = \min\{RC_G(q)\colon q \in Q\}.$$

Based on Proposition 3.1, initially, Arc contains the arcs of the paths in Q with that minimum robustness cost, i.e.

$$Arc = \Big\{(i,j) \in A(q)\colon q \in Q \text{ and } RC_G(q) = RCmin\Big\}.$$

When an arc (i,j) is selected in Arc to be scanned, one must check if node n is reachable from node 1 in the network $G^{-}_{(i,j)}$. In that case, the shortest $(1,n)$-paths in $G^{-}_{(i,j)}$ for the scenarios s_u, $p^{1,s_u}(G^{-}_{(i,j)})$, with $u \in U(i,j)$, have to be determined. The inequality (3.2) has to be tested for those scenarios and the algorithm can halt the analysis of (i,j) when the inequality is satisfied.

For all the algorithms, list A_1 is used to collect the robust 1-persistent arcs. For a given scenario, determining the associate shortest $(1,n)$-path of G or of $G^{-}_{(i,j)}$, for some arc (i,j), and its cost, can be done by applying any shortest path algorithm.

Static approach This method sets the initial $RCmin$ as given above to test condition (3.2) for all the arcs in Arc along the process. This value remains unchanged along the algorithm. The pseudo-code that summarizes the procedure for determining the robust 1-persistent arcs of G is presented in Algorithm 4.

Algorithm 4: Static version for finding robust 1-persistent arcs

```
1  Q ← ∅;
2  for u ∈ U_k do
3      Compute p^{1,s_u}(G); Q ← Q ∪ {p^{1,s_u}(G)};
4      LB_{1n}^{s_u}(G) ← c_G^{s_u}(p^{1,s_u}(G));
5  RCmin ← min{RC_G(q): q ∈ Q};
6  Arc ← {(i,j) ∈ A(q): q ∈ Q and RC_G(q) = RCmin};
7  A_1 ← ∅;
8  while Arc ≠ ∅ do
9      Choose an arc (i,j) ∈ Arc; Arc ← Arc\{(i,j)};
10     if node n is reachable from node 1 in G^-_{(i,j)} then
11         U(i,j) ← {u ∈ U_k: (i,j) ∈ p^{1,s_u}(G)};
12         for u ∈ U(i,j) do
13             Compute p^{1,s_u}(G^-_{(i,j)});
14             RDA^u_{(i,j)} ← c_G^{s_u}(p^{1,s_u}(G^-_{(i,j)})) − LB_{1n}^{s_u}(G);
15             if RDA^u_{(i,j)} > RCmin then
16                 A_1 ← A_1 ∪ {(i,j)}; break;
17 return A_1;
```

Computational time complexity order Two phases of Algorithm 4 should be considered. The first corresponds to determining the costs $LB_{1n}^{s_u}(G)$, $u \in U_k$, and the robustness cost, $RCmin$. As explained in Section 2.2, this procedure is of $O_1^a = \mathcal{O}(km + k^2n)$ for acyclic networks and of $O_1^c = \mathcal{O}(k(m + n\log n) + k^2n)$ for general networks.

The second phase concerns the analysis of the arcs in Arc. This set only contains arcs of the shortest $(1,n)$-paths, $p^{1,s_u}(G)$, $u \in U_k$, each having $n-1$ arcs at most. Hence, Arc has $k(n-1)$ elements at most. For each arc (i,j) selected in Arc and each scenario s_u, for $u \in U(i,j)$, which has at most k elements, the computation of the shortest $(1,n)$-path $p^{1,s_u}(G^-_{(i,j)})$ and its cost $LB_{1n}^{s_u}(G^-_{(i,j)})$ are required. They are performed in $\mathcal{O}(m+n) = \mathcal{O}(m)$, for acyclic networks, and in $\mathcal{O}(m+n\log n)$ for general networks, as seen in Section 2.2, when considering only one scenario for the costs. Calculating $RD_G^{s_u}(p^{1,s_u}(G^-_{(i,j)}))$ and applying test (3.2) requires $\mathcal{O}(1)$ time. Therefore, one obtains $O_2^a = \mathcal{O}(k^2mn)$ time for acyclic networks and $O_2^c = \mathcal{O}(k^2n(m + n\log n))$ time for general networks.

In conclusion, Algorithm 4 is polynomial in time and it has a complexity of $O_1^a + O_2^a = \mathcal{O}(k^2mn)$ for acyclic networks, and of $O_1^c + O_2^c = \mathcal{O}(k^2mn + k^2n^2\log n)$ for general networks.

Example In the following, the application of Algorithm 4 is exemplified for finding robust 1-persistent arcs in the network $G_4 = G_4(V, A, \{1, 2\})$ represented in Figure 3.2.

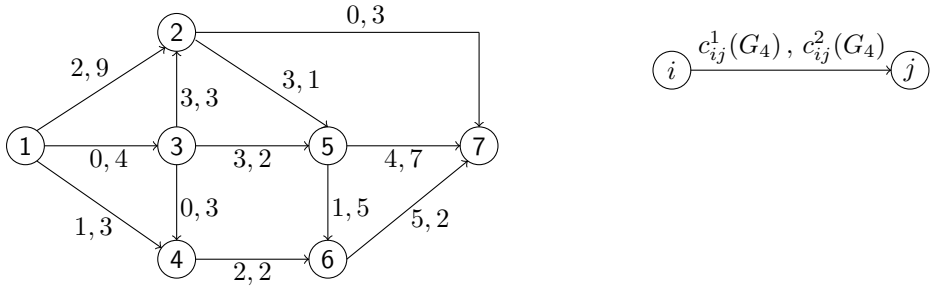
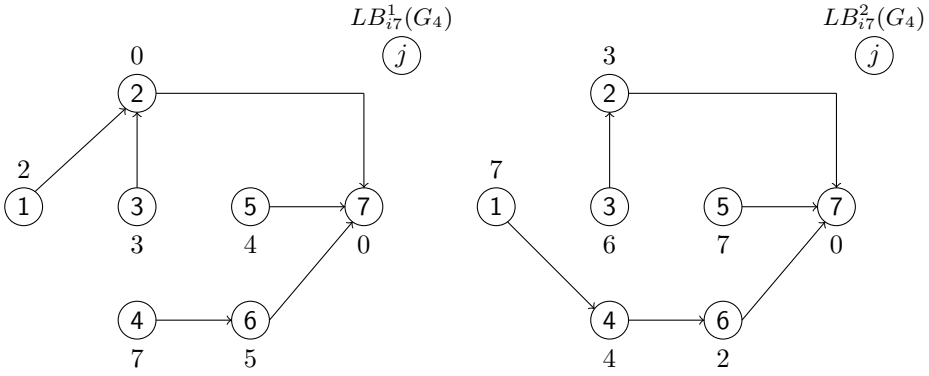


Figure 3.2: Network G_4



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

Figure 3.3: Shortest path trees rooted at $n = 7$ in G_4

Figure 3.3 shows the trees of the shortest paths rooted at node 7 in G_4, $\mathcal{T}^1_7(G_4)$ and $\mathcal{T}^2_7(G_4)$. After the trees calculation, the list Q is set to $\{p^{1,1}(G_4), p^{1,2}(G_4)\}$, with $p^{1,1}(G_4) = \langle 1, 2, 7\rangle$, $LB^1_{17}(G_4) = 2$, and $p^{1,2}(G_4) = \langle 1, 4, 6, 7\rangle$, $LB^2_{17}(G_4) = 7$. The robustness costs of these paths are $RC_{G_4}(p^{1,1}(G_4)) = 5$ and $RC_{G_4}(p^{1,2}(G_4)) = 6$. Hence, $p^{1,1}(G_4)$ is the best path in Q, and, therefore, the static algorithm starts by setting

$$Arc = \{(1, 2), (2, 7)\} \quad \text{and} \quad RCmin = 5.$$

The latter value is used in every test of (3.2) and it does not change along the process. First, the arc $(1, 2)$ is considered and $U(1, 2) = \{1\}$. Node 7 is reachable from node 1 in $(G_4)^-_{(1,2)}$ and $p^{1,1}((G_4)^-_{(1,2)}) = \langle 1, 3, 2, 7\rangle$, with

$$RDA^1_{(1,2)} = c^1_{G_4}(\langle 1, 3, 2, 7\rangle) - LB^1_{17}(G_4) = 1 \leq RCmin.$$

Therefore, condition (3.2) is not satisfied, and nothing can be concluded about arc $(1,2)$. Afterwards, arc $(2,7)$ is selected and $U(2,7) = \{1\}$. Now, node 7 is reachable from node 1 in $(G_4)^-_{(2,7)}$ and $p^{1,1}((G_4)^-_{(2,7)}) = \langle 1,3,5,7\rangle$, with

$$RDA^1_{(2,7)} = c^1_{G_4}(\langle 1,3,5,7\rangle) - LB^1_{17}(G_4) = 5 \leq RCmin.$$

Once again, condition (3.2) is not satisfied, thus, Algorithm 4 finishes without having detected any robust 1-persistent arc, i.e.

$$A_1 = \emptyset.$$

Consequently, preprocessing robust 1-persistent arcs with the static method is not effective for this example.

Dynamic approach For this version, the value of variable $RCmin$ may change along the algorithm. The $(1,n)$-paths computed by the algorithm are stored in a list X_P, without repetitions. The set of arcs Arc to scan may also change, every time a new $(1,n)$-path q satisfies $RC_G(q) \leq RCmin$. Under this condition, Proposition 3.1 allows to test the arcs shared by path q and by the shortest $(1,n)$-paths $p^{1,s_u}(G)$, $u \in U_k$, which were not identified previously as robust 1-persistent. In case $RC_G(q) = RCmin$, those arcs are included in set Arc for scanning, while, if $RC_G(q) < RCmin$, those arcs update Arc, since path q is a new candidate for the optimal solution. Hence, one can write

$$Arc = \begin{cases} Arc \cup \left[\left(A(q) \cap \left(\bigcup_{u=1}^{k} A(p^{1,s_u}(G))\right)\right) \backslash A_1\right] & \text{if} \quad RC_G(q) = RCmin \\ \left(A(q) \cap \left(\bigcup_{u=1}^{k} A(p^{1,s_u}(G))\right)\right) \backslash A_1 & \text{if} \quad RC_G(q) < RCmin \end{cases}$$

For a selected $(i,j) \in Arc$, path q takes the particular form

$$q = p^{1,s_u}(G^-_{(i,j)}),\ u \in U(i,j).$$

Whenever $RC_G(q) < RCmin$, $RCmin$ is updated to $RC_G(q)$. Some arcs may be scanned more than once, because the analyzed $(1,n)$-paths may have arcs in common. This makes that some tests may be repeated after $RCmin$ is updated. Hence, the variables $RDA^u_{(i,j)}$, $(i,j) \in Arc$, $u \in U(i,j)$, are used to store the path robust deviations. A list X_A is used to store the arcs that have already been analyzed along the process. The dynamic procedure for identifying robust 1-persistent arcs is outlined in Algorithm 5.

Algorithm 5: Dynamic version for finding robust 1-persistent arcs

$Q \leftarrow \emptyset$;
for $u \in U_k$ **do**
  Compute $p^{1,s_u}(G)$; $Q \leftarrow Q \cup \{p^{1,s_u}(G)\}$;
  $LB_{1n}^{s_u}(G) \leftarrow c_G^{s_u}(p^{1,s_u}(G))$;
$RCmin \leftarrow \min\{RC_G(q) \colon q \in Q\}$;
$Arc \leftarrow \{(i,j) \in A(q) \colon q \in Q \text{ and } RC_G(q) = RCmin\}$;
$X_P \leftarrow Q\,; X_A \leftarrow \emptyset\,; A_1 \leftarrow \emptyset$;
while $Arc \neq \emptyset$ **do**
  Choose an arc $(i,j) \in Arc$; $Arc \leftarrow Arc \backslash \{(i,j)\}$;
  if $(i,j) \notin X_A$ *and node* n *is reachable from node 1 in* $G^-_{(i,j)}$ **then**
    $X_A \leftarrow X_A \cup \{(i,j)\}$;
    $U(i,j) \leftarrow \{u \in U_k \colon (i,j) \in p^{1,s_u}(G)\}$;
    for $u \in U(i,j)$ **do**
      Compute $p^{1,s_u}(G^-_{(i,j)})$; $q \leftarrow p^{1,s_u}(G^-_{(i,j)})$;
      $RDA^u_{(i,j)} \leftarrow c_G^{s_u}(q) - LB_{1n}^{s_u}(G)$;
      if $RDA^u_{(i,j)} > RCmin$ **then**
        $A_1 \leftarrow A_1 \cup \{(i,j)\}$; **break**;
      if $q \notin X_P$ **then**
        $X_P \leftarrow X_P \cup \{q\}$;
        $RC_G(q) \leftarrow \max\left\{RDA^u_{(i,j)}, \max\left\{RD_G^{s_{u'}}(q) : u' \in U_k \backslash \{u\}\right\}\right\}$;
        if $RC_G(q) = RCmin$ **then** $Arc \leftarrow Arc \cup \left[\left(A(q) \cap \left(\bigcup_{u'=1}^{k} A(p^{1,s_{u'}}(G))\right)\right) \backslash A_1\right]$;
        if $RC_G(q) < RCmin$ **then**
          $RCmin \leftarrow RC_G(q)$; $Arc \leftarrow \left(A(q) \cap \left(\bigcup_{u'=1}^{k} A(p^{1,s_{u'}}(G))\right)\right) \backslash A_1$;
  else
    for $u \in U(i,j)$ **do**
      if $RDA^u_{(i,j)} > RCmin$ **then**
        $A_1 \leftarrow A_1 \cup \{(i,j)\}$; **break**;
return A_1;

Computational time complexity order Algorithm 5 performs three additional tasks, when compared to Algorithm 4. They are the calculation of the robustness costs of the $(1,n)$-paths $p^{1,s_u}(G^-_{(i,j)})$, $(i,j) \in Arc$, $u \in U(i,j)$, the updates of set Arc, and the repetition of the tests (3.2) after updating $RCmin$.

For the first task, assuming the costs $LB^{s_u}_{1n}(G)$, $u \in U_k$, were previously computed, the robustness cost of $p^{1,s_u}(G^-_{(i,j)})$, $(i,j) \in Arc$, $u \in U(i,j)$, in G is obtained in $\mathcal{O}(kn)$ time.

In what concerns the second procedure, the set Arc is updated by means of intersections, unions and differences of subsets in $\bigcup_{u=1}^{k} A(p^{1,s_u}(G))$ and set $A(q)$, which have $(k+1)(n-1)$ arcs at most. An efficient way to make such operations is with indexation using hash sets [8], 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, repeating the test (3.2), for a given arc $(i,j) \in Arc$, requires $\mathcal{O}(1)$ operations for each scenario s_u, with $u \in U(i,j)$, because $RDA^u_{(i,j)}$ was already determined.

In a worst case, the tasks described above are performed up to $k^2(n-1)$ times, one per each scenario in s_u, $u \in U_k$, and each arc selected in Arc, with up to $k(n-1)$ elements. Consequently, the dynamic approach increases by $\mathcal{O}(k^3n^2)$ the number of operations performed by Algorithm 4.

Therefore, Algorithm 5 has a time complexity of $\mathcal{O}(k^2mn + k^3n^2)$ for acyclic networks and of $\mathcal{O}(k^2mn + k^2n^2 \log n + k^3n^2)$ for general networks.

Because Algorithms 4 and 5 start by using the same $RCmin$, it should be noted that all the robust 1-persistent arcs identified with the former version are still identified with the latter. In fact, let $RCmin_{Ini} = \min\{RC_G(q) : q \in Q\}$ be the initial cost lower-bound for both methods. Then, every robust 1-persistent arc (i,j) detected by the static algorithm, satisfies $RD^{s_u}_G(p^{1,s_u}(G^-_{(i,j)})) > RCmin_{Ini}$, for some $u \in U(i,j)$. Since every $(1,n)$-path q' obtained in the dynamic algorithm that sets a minimal robustness cost, satisfies $RC_G(q) \leq RCmin_{Ini}$, then $(i,j) \in A(q)$. Otherwise, the definition of robustness cost and the fact that $p^{1,s_u}(G^-_{(i,j)})$ is the shortest $(1,n)$-path in G not containing arc (i,j), imply that $RC_G(q') \geq RD^{s_u}_G(q') \geq RD^{s_u}_G(p^{1,s_u}(G^-_{(i,j)})) > RCmin_{Ini}$, which contradicts the latter condition. Consequently, the arcs of $A(q') \cap \left(\bigcup_{u=1}^{k} A(p^{1,s_u}(G)) \right)$, which update or are included in set Arc of Algorithm 5, contain necessarily arc (i,j). Therefore, this arc is scanned in the dynamic version, being identified as robust 1-persistent by a cost lower-bound not greater than $RCmin_{Ini}$.

Example Next, Algorithm 5 is applied to preprocess robust 1-persistent arcs in network G_4 of Figure 3.2. With that goal in mind, the trees $\mathcal{T}^1_7(G_4)$ and $\mathcal{T}^2_7(G_4)$ in Figure 3.3 are taken into consideration.

Analogously to Algorithm 4, the dynamic method starts with

$$Arc = \{(1,2),(2,7)\} \quad \text{and} \quad RCmin = 5,$$

and selects arc $(1,2)$ to scan. Condition (3.2) is not satisfied in this case, because $RDA^1_{(1,2)} = 1$. Additionally, the robustness cost of $p^{1,1}((G_4)^-_{(1,2)}) = \langle 1,3,2,7\rangle$ in G_4 is determined by

$$RC_{G_4}(\langle 1,3,2,7\rangle) = \max\left\{1, RDA^2_{(1,2)}\right\} = \max\left\{1, c^2_{G_4}(\langle 1,3,2,7\rangle) - LB^2_{17}(G_4)\right\} = 3,$$

which allows to improve $RCmin$ to

$$RCmin = 3.$$

Therefore, because $A_1 = \emptyset$, Arc is updated to

$$Arc = A(\langle 1,3,2,7\rangle) \cap \left(\bigcup_{u=1}^{2} A(p^{1,u}(G_4))\right) = \{(2,7)\}.$$

Then, arc $(2,7)$ is selected, with $p^{1,1}((G_4)^-_{(2,7)}) = \langle 1,3,5,7\rangle$ satisfying

$$RDA^1_{(2,7)} = 5 > RCmin,$$

from the calculations for the static version. This means condition (3.2) holds, and, consequently, arc $(2,7)$ is robust 1-persistent, i.e.

$$A_1 = \{(2,7)\}.$$

Computing a robust shortest path after preprocessing After running Algorithm 5, it is concluded that arc $(2,7)$ must be contained in the optimal solution, since it is robust 1-persistent. Thus, the reduced network is obtained by removing from G_4 all its remaining arcs that start in node 2, $(2,5)$, or that end in node 7, $(5,7)$ and $(6,7)$. Figure 3.4 depicts the resultant network. Arc $(2,7)$ is marked with a thick line and the nodes and arcs which cannot be included in any optimal $(1,7)$-path are marked with a dashed line.

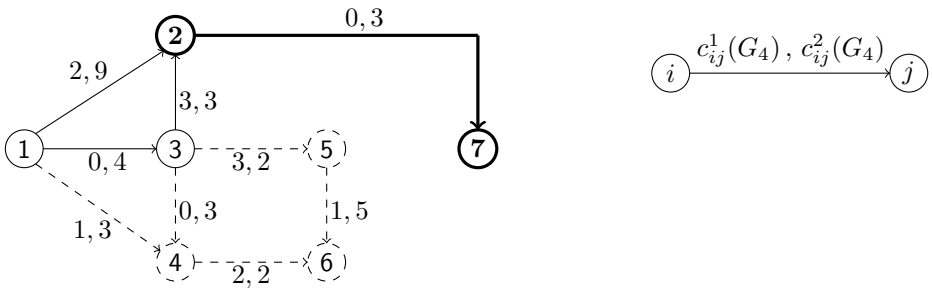


Figure 3.4: Reduced network of G_4 after preprocessing robust 1-persistent arcs with Algorithm 5

There are only two $(1,7)$-paths in the reduced network, in which arc $(2,7)$ is included. They are $p^{1,1}(G_4) = \langle 1,2,7\rangle$, with $RC_{G_4}(p^{1,1}(G_4)) = 5$, and $q = \langle 1,3,2,7\rangle$, with $RC_{G_4}(q) = 3$. Therefore, q is the robust shortest path in G_4.

It should be noted that for the same example, the dynamic method was more effective than the static method on preprocessing robust 1-persistent arcs. In fact, with Algorithm 4, no robust 1-persistent arcs were identified, making this strategy useless to facilitate the determination of a robust shortest path. On the contrary, with Algorithm 5, one robust 1-persistent arc was detected, which significantly reduced the number of potentially optimal paths and, therefore, the effort invested on solving the problem.

3.3 Identification of robust 0-persistent nodes

Karasan et al. [26] addressed the robust shortest path problem with interval data and introduced preprocessing techniques to identify robust 0-persistent arcs in layered networks. Later, Catanzaro et al. [13] developed a similar idea for the same problem and extended the previous results to networks that may contain cycles. Besides detecting robust 0-persistent arcs, the new rules detected robust 0-persistent nodes as well, and, consequently, the size of the problem could be reduced further.

Both results are based on the shortest $(1,n)$-paths for scenarios that result from the lower and the upper-limits of the cost intervals. In particular, the first result identifies the arcs that are not part of any shortest $(1,n)$-path under the scenario that attributes the lower-limits of the cost intervals to the arcs of that path and the upper-limits of the cost intervals to the remaining arcs of the network. The second result evaluates the nodes which do not belong to the shortest $(1,n)$-path under the scenario associated with the upper-limits of the cost intervals. A node i is robust 0-persistent when the cost of that path is smaller than the cost of the path that results from the concatenation of the shortest $(1,i)$-path with the shortest (i,n)-path under the scenario associated with the lower-limits of the cost intervals.

For the finite multi-scenario model, the identification of robust 0-persistent nodes is motivated by the latter result. In the following, a sufficient condition is provided with the same purpose, when a $(1,n)$-path and its robustness cost in G are known by hypothesis. The result is based on the robust deviations of the paths obtained from the concatenation described above, taking into account all scenarios.

Proposition 3.2. *Consider a path $q \in P_{1n}(G)$, and a node $i \notin V(q)$. If*

$$\exists u' \in U_k : \; RD_G^{s_{u'}}(p_{1i}^{1,s_{u'}}(G) \diamond p_{in}^{1,s_{u'}}(G)) > RC_G(q),$$

then node i is robust 0-persistent.

Proof. Let $i \in V \backslash V(q)$ and q' be any path in $P_{1n}(G)\backslash\{q\}$ such that $i \in V(q')$. Let ${q'}_{1i}$ and ${q'}_{in}$ represent the $(1,i)$-sub-path and the (i,n)-sub-path of q', respectively. By definition of robustness cost of a $(1,n)$-path in G,

$$RC_G(q') = \max_{u \in U_k} RD_G^{s_u}(q') = \max_{u \in U_k} \Big\{ c_G^{s_u}({q'}_{1i}) + c_G^{s_u}({q'}_{in}) - LB_{1n}^{s_u}(G) \Big\}.$$

Given that $p_{1i}^{s_u}(G)$ and $p_{in}^{s_u}(G)$ are the shortest $(1,i)$-path and the shortest (i,n)-path in scenario s_u, $u \in U_k$, in G, respectively, then

$$RC_G(q') \geq \max_{u \in U_k} \Big\{ LB_{1i}^{s_u}(G) + LB_{in}^{s_u}(G) - LB_{1n}^{s_u}(G) \Big\} = \max_{u \in U_k} RD_G^{s_u}(p_{1i}^{s_u}(G) \diamond p_{in}^{s_u}(G)).$$

Consequently, if $RD_G^{s_{u'}}(p_{1i}^{1,s_{u'}}(G) \diamond p_{in}^{1,s_{u'}}(G)) > RC_G(q)$ is satisfied for some $u' \in U_k$, then

$$\max_{u \in U_k} RD_G^{s_u}(p_{1i}^{s_u}(G) \diamond p_{in}^{s_u}(G)) > RC_G(q).$$

Therefore,

$$RC_G(q') > RC_G(q),$$

which means that any path in G that contains node i cannot be a robust shortest path. Therefore, node i is robust 0-persistent. □

An alternative strategy to find robust 0-persistent nodes can be derived when a robust 1-persistent arc is identified in G. This technique is able to detect robust 0-persistent nodes more easily than the result above, because it avoids the calculation of path robust deviations. The new result, given in the following, follows from the combination of Propositions 3.1 and 3.2.

Corollary 3.3. *Let (i,j) be a robust 1-persistent arc and $q \in P_{1n}(G)$ be a path, such that $(i,j) \in A(q) \cap \Big(\bigcup_{u=1}^{k} A(p^{1,s_u}(G)) \Big)$. Then, any node $j' \notin V(q)$ such that $(i,j) \notin p_{1j'}^{1,s_{u'}}(G)$ and $(i,j) \notin p_{j'n}^{1,s_{u'}}(G)$, for some $u' \in U(i,j)$, is robust 0-persistent.*

Proof. If arc (i,j) is robust 1-persistent and $(i,j) \in A(q) \cap \Big(\bigcup_{u=1}^{k} A(p^{1,s_u}(G)) \Big)$ for some $q \in P_{1n}(G)$, then, according to Proposition 3.1, there exists some $u' \in U(i,j)$, for which

$$RD_G^{s_{u'}}(p^{1,s_{u'}}(G_{(i,j)}^-)) > RC_G(q).$$

Since $p^{1,s_{u'}}(G_{(i,j)}^-)$ is the shortest $(1,n)$-path in G that does not contain arc (i,j) in scenario $s_{u'}$, then, any other path $q' \in P_{1n}(G)$, such that $(i,j) \notin A(q')$, satisfies $c_G^{s_{u'}}(q') \geq c_G^{s_{u'}}(p^{1,s_{u'}}(G_{(i,j)}^-))$. From the condition above,

$$RD_G^{s_{u'}}(q') \geq RD_G^{s_{u'}}(p^{1,s_{u'}}(G_{(i,j)}^-)) > RC_G(q).$$

Consequently, any node $j' \notin V(q)$, such that $(i,j) \notin p_{1j'}^{1,s_{u'}}(G)$ and $(i,j) \notin p_{j'n}^{1,s_{u'}}(G)$ makes that $p_{1j'}^{1,s_{u'}}(G) \diamond p_{j'n}^{1,s_{u'}}(G)$ does not contain arc (i,j), and, therefore, q' can be set to that $(1,n)$-path. From this fact and Proposition 3.2, one concludes that j' is a robust 0-persistent node. □

Since the search for robust 1-persistent arcs is restricted to the set of arcs of the shortest $(1,n)$-paths for the scenarios of the model, their identification can be compromised for networks with few arcs of those arcs in comparison with the total number of arcs. This fact is dependent on the density of the network and also on the number of scenarios of the model. Because of this limitation, techniques to deal with the rule for preprocessing robust 0-persistent nodes are presented. The possibility of detecting the highest possible number of robust 0-persistent nodes with the least possible computational effort is related with two aspects concerned with the condition of Proposition 3.2. One is the number of involved scenarios and the other is how the lower-bound decreases along the process. In order to simplify notation, let $RCmin$ be the variable which sets the lower-bound, Nod denote the set of nodes to be scanned and

$$RDV_i^u = RD_G^{s_u}(p_{1i}^{1,s_u}(G) \diamond p_{in}^{1,s_u}(G)) \,,\; i \in V \,,\; u \in U_k,$$

represent the path robust deviations. Then, Proposition 3.2 can be rewritten, considering that, for any node $i \in Nod$, if

$$\exists u' \in U_k \;:\; RDV_i^{u'} > RCmin, \tag{3.3}$$

is satisfied, then node i is robust 0-persistent.

In the following, the static and the dynamic algorithmic approaches to identify robust 0-persistent nodes will be introduced, adapted to the new notation.

The number of scenarios used to test condition (3.3) may make the robust 0-persistent nodes test computationally demanding. In order to make this task lighter, the number of considered scenarios can be restricted. This can be done by imposing M, $M \leq k$, as the largest scenario index used in the tests. Moreover, for each node $i \in Nod$, when the first scenario index u_i, $u_i \in U_k$, for which (3.3) holds is known, then i is a robust 0-persistent node and its analysis can halt. Hence, the tests for scenarios $s_{u_i+1}, \ldots, s_M$ can be skipped. Generally, if $u_i < M$, $i \in Nod$, the computation of the trees $\mathcal{T}_1^{s_u}(G)$ can be skipped for any $u = \max\{u_i : i \in Nod\}+1, \ldots, M$.

For all the algorithms, V_0 is the list which collects the robust 0-persistent nodes.

Static approach The variable $RCmin$ is initialized and set like for Algorithm 4. Since this value is related with the minimum robustness cost of the distinct shortest $(1,n)$-paths in list Q, Proposition 3.2 allows to initialize

$$Nod = V \backslash \{i \in V(q) \colon q \in Q \text{ and } RC_G(q) = RCmin\}.$$

Then, all the nodes in Nod are scanned by testing (3.3) for the initial $RCmin$. The pseudo-code that summarizes the static version of the algorithm to determine robust 0-persistent nodes is given in Algorithm 6.

Algorithm 6: Static version for finding robust 0-persistent nodes, given M

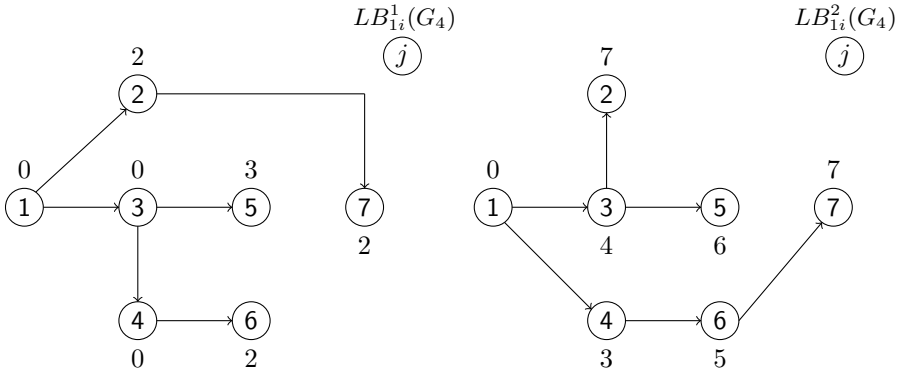
```
Q ← ∅;
for u ∈ U_k do
    Compute T_n^{s_u}(G); Q ← Q ∪ {p^{1,s_u}(G)};
    for i = 1, ..., n − 1 do LB_in^{s_u}(G) ← c_G^{s_u}(p_in^{1,s_u}(G));
RCmin ← min{RC_G(q): q ∈ Q};
Nod ← V\{i ∈ V(q): q ∈ Q and RC_G(q) = RCmin};
V_0 ← ∅;
while Nod ≠ ∅ do
    Choose a node i ∈ Nod; Nod ← Nod\{i};
    for u = 1, ..., M do
        if T_1^{s_u}(G) was not determined yet then
            Compute T_1^{s_u}(G);
            for i = 2, ..., n − 1 do LB_1i^{s_u}(G) ← c_G^{s_u}(p_1i^{1,s_u}(G));
        RDV_i^u ← LB_1i^{s_u}(G) + LB_in^{s_u}(G) − LB_1n^{s_u}(G);
        if RDV_i^u > RCmin then
            V_0 ← V_0 ∪ {i}; break;
return V_0;
```

Computational time complexity order In the worst case, Algorithm 6 has the same time complexity order as Algorithm 4 to determine $RCmin$. Then, the second phase is dedicated to the search for robust 0-persistent nodes. The computation of the tree $\mathcal{T}_1^{s_u}(G)$ and of the costs $LB_{1i}^{s_u}(G)$ is similar to the computation of $\mathcal{T}_n^{s_u}(G)$ and $LB_{in}^{s_u}(G)$, $i \in V$, $u \in U_k$. Since such procedure only requires the scenario where the paths are the shortest to calculate their costs, it has time of $\mathcal{O}(m)$ for acyclic networks and of $\mathcal{O}(m + n \log n)$ for general networks. Then, for each node selected in $Nod \subseteq V\backslash\{1, n\}$, condition (3.3) is checked in $\mathcal{O}(1)$ time, which means $\mathcal{O}(n)$ operations are required. Since the analysis of the nodes considers at most k scenarios, the second phase is performed in $O_2^a = \mathcal{O}(k(m+n)) = \mathcal{O}(km)$ time for acyclic networks and in $O_2^c = \mathcal{O}(k(m + n \log n))$ time for general networks.

Consequently, Algorithm 6 has a time complexity of $O_1^a + O_2^a = \mathcal{O}(km + k^2 n)$ for acyclic networks and of $O_1^c + O_2^c = \mathcal{O}(km + kn \log n + k^2 n)$ for general networks.

Example The application of Algorithm 6 is now illustrated for identifying robust 0-persistent nodes in network G_4 of Figure 3.2. The trees $\mathcal{T}_7^1(G_4)$ and $\mathcal{T}_7^2(G_4)$ are given in

Figure 3.3 and the homologous trees rooted at node 1, $\mathcal{T}_1^1(G_4)$ and $\mathcal{T}_1^2(G_4)$, are shown in Figure 3.5.



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

Figure 3.5: Shortest path trees rooted at node 1 in G_4

In what follows, the number of scenarios tested in (3.3) is limited to $M \in \{1, 2\}$. Because $p^{1,1}(G_4) = \langle 1, 2, 7\rangle$ is the shortest $(1, 7)$-path with the least robustness cost, 5, Algorithm 6 starts with

$$Nod = \{3, 4, 5, 6\} \quad \text{and} \quad RCmin = 5.$$

The latter value is set for the tests on (3.3), applied to all the nodes in Nod along the algorithm.

- $M = 1$

  Starting with node 3, the inequality (3.3) is not satisfied in scenario 1, given that

  $$RDV_3^1 = LB_{13}^1(G_4) + LB_{37}^1(G_4) - LB_{17}^1(G_4) = 1 \leq RCmin.$$

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

  $$RDV_i^1 = LB_{1i}^1(G_4) + LB_{i7}^1(G_4) - LB_{17}^1(G_4) = 5 \leq RCmin\,,\ i = 4, 5, 6.$$

  Therefore, no robust 0-persistent nodes are detected when considering only scenario 1, i.e.

  $$V_0 = \emptyset.$$

- $M = 2$

  For scenario 2, the nodes 3, 4 and 6 still do not satisfy (3.3), given that

  $$RDV_3^2 = LB_{13}^2(G_4) + LB_{37}^2(G_4) - LB_{17}^2(G_4) = 3 \leq RCmin,$$

and

$$RDV_i^2 = LB_{1i}^2(G_4) + LB_{i7}^2(G_4) - LB_{17}^2(G_4) = 0 \leq RCmin\,,\ i = 4, 6.$$

Nevertheless, (3.3) holds for node 5 and scenario 2,

$$RDV_5^2 = LB_{15}^2(G_4) + LB_{57}^2(G_4) - LB_{17}^2(G_4) = 6 > RCmin,$$

therefore, node 5 is the only one identified as robust 0-persistent, i.e.

$$V_0 = \{5\}.$$

Computing a robust shortest path after preprocessing Since node 5 was the only node identified as robust 0-persistent, the arcs that start in node 5, $(5, 6)$ and $(5, 7)$, or that end in node 5, $(2, 5)$ and $(3, 5)$, are removed from G_4. The reduced network obtained is represented in Figure 3.6.

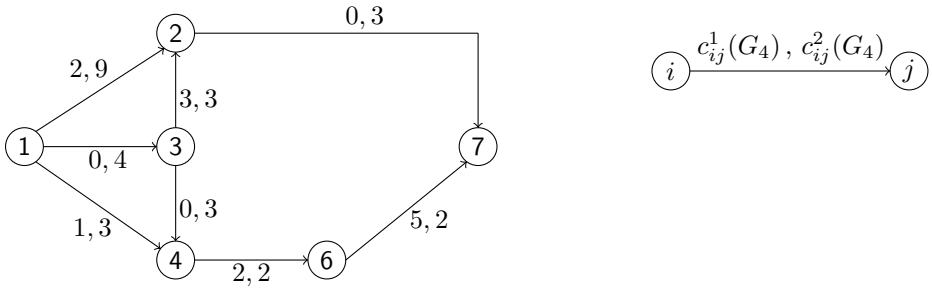


Figure 3.6: Reduced network of G_4 after preprocessing robust 0-persistent nodes with Algorithm 6

There are four $(1, 7)$-paths in the reduced network, $p^{1,1}(G_4) = \langle 1, 2, 7\rangle$, $p^{1,2}(G_4) = \langle 1, 4, 6, 7\rangle$, $q = \langle 1, 3, 2, 7\rangle$ and $q' = \langle 1, 3, 4, 6, 7\rangle$, with robustness costs of 5, 6, 3 and 5, respectively. Consequently, q is the robust shortest path in G_4, as concluded in Section 3.1.

Dynamic approach In this algorithm, in order to spare computational effort, instead of initializing $RCmin$ like for the previous methods, only path $p^{1,s_1}(G)$ is considered to set the initial $RCmin$ and Nod. Then, the algorithm starts with

$$RCmin = RC_G(p^{1,s_1}(G)) \quad \text{and} \quad Nod = V \backslash V(p^{1,s_1}(G)).$$

The variables $RCmin$ and Nod are updated along the method, according to the robustness cost in G of the computed $(1, n)$-paths. Analogously to Algorithm 5, list X_P stores these paths without repetitions. Whenever a computed $(1, n)$-path q satisfies $RC_G(q) \leq RCmin$, set Nod is updated as explained next.

When searching for robust 0-persistent nodes, Proposition 3.2 establishes that the analysis of the nodes of path q, $V(q)$, can be skipped. Thus, if $RC_G(q) = RCmin$, the nodes of $V(q)$ can be removed from Nod, and, if $RC_G(q) < RCmin$, the search focuses all the nodes outside $V(q)$ that have not already been identified as robust 0-persistent. Then,

$$Nod = \begin{cases} Nod \backslash V(q) & \text{if} \quad RC_G(q) = RCmin \\ V \backslash (V(q) \cup V_0) & \text{if} \quad RC_G(q) < RCmin \end{cases}$$

For a selected node $i \in Nod$, path q has the particular form

$$q = p_{1i}^{1,s_u}(G) \diamond p_{in}^{1,s_u}(G)\,,\ u \in U_k.$$

Whenever $RC_G(q) < RCmin$, $RCmin$ is updated to $RC_G(q)$. As the $(1,n)$-paths under analysis may have nodes in common, the scanning of some can be repeated after $RCmin$ is updated. In order to avoid repeating the calculation of previous path robust deviations, the variables RDV_i^u, $i \in Nod$, $u \in U_k$, are used to store such values. In addition, a list X_V is used to store the nodes that have been previously scanned. The pseudo-code is given in Algorithm 7.

Computational time complexity order In terms of the worst case computational time complexity, the first phases of Algorithms 6 and 7 are similar. The latter initializes $RCmin$ by $RC_G(p^{1,s_1}(G))$ only, which means it is performed in $O_1^a = \mathcal{O}(km + kn) = \mathcal{O}(km)$ time for acyclic networks and in $O_1^c = \mathcal{O}(k(m + n\log n))$ for general networks.

The second phase concerns searching for robust 0-persistent nodes, which compared to the static version has the additional work of calculating RDV_i^u, $i \in Nod$, $u \in U_k$, updating set Nod, and repeating the tests (3.3) due to the updates of $RCmin$.

For the first task, assuming that the trees $\mathcal{T}_n^{s_u}(G)$ and $\mathcal{T}_1^{s_u}(G)$, $u \in U_k$, and the associate costs for all scenarios were previously computed, RDV_i^u, $i \in Nod$, $u \in U_k$, is obtained in $\mathcal{O}(k)$ time. The second task concerns the update of Nod and involves differences and unions of sets with n nodes at most. These operations require an $\mathcal{O}(n)$ complexity, when using indexation by hash sets [8]. The third procedure demands $\mathcal{O}(1)$ operations for each scenario s_u, $u \in U_k$, and each node $i \in Nod$, since RDV_i^u was already determined.

In a worst case, the three tasks above are performed $k(n-2)$ times at most, one per each scenario s_u, $u \in U_k$, and each selected node in Nod, with up to $n-2$ nodes. Thus, an additional work of $\mathcal{O}(kn^2 + k^2n)$ is added to the second phase of the static version.

In conclusion, Algorithm 7 has a time complexity of $\mathcal{O}(kn^2 + k^2n)$ for all types of networks, since $\log n \ll n$ and $m < n^2$.

Algorithm 7: Dynamic version for finding robust 0-persistent nodes, given M

for $u \in U_k$ **do**
  Compute $\mathcal{T}_n^{s_u}(G)$;
  for $i = 1, \ldots, n-1$ **do** $LB_{in}^{s_u}(G) \leftarrow c_G^{s_u}(p_{in}^{1,s_u}(G))$;
$RCmin \leftarrow RC_G(p^{1,s_1}(G))$; $Nod \leftarrow V \backslash V(p^{1,s_1}(G))$;
$X_P \leftarrow \{p^{1,s_1}(G)\}$; $X_V \leftarrow \emptyset$; $V_0 \leftarrow \emptyset$;
while $Nod \neq \emptyset$ **do**
  Choose a node $i \in Nod$; $Nod \leftarrow Nod \backslash \{i\}$;
  if $i \notin X_V$ **then**
    $X_V \leftarrow X_V \cup \{i\}$;
    for $u = 1, \ldots, M$ **do**
      if $\mathcal{T}_1^{s_u}(G)$ *was not determined yet* **then**
        Compute $\mathcal{T}_1^{s_u}(G)$;
        for $i = 2, \ldots, n-1$ **do** $LB_{1i}^{s_u}(G) \leftarrow c_G^{s_u}(p_{1i}^{1,s_u}(G))$;
      $RDV_i^u \leftarrow LB_{1i}^{s_u}(G) + LB_{in}^{s_u}(G) - LB_{1n}^{s_u}(G)$;
      if $RDV_i^u > RCmin$ **then**
        $V_0 \leftarrow V_0 \cup \{i\}$; **break**;
      $q \leftarrow p_{1i}^{1,s_u}(G) \diamond p_{in}^{1,s_u}(G)$;
      if $q \notin X_P$ **then**
        $X_P \leftarrow X_P \cup \{q\}$;
        $RC_G(q) \leftarrow \max\left\{RDV_i^u, \max\left\{RD_G^{s_{u'}}(q) : u' \in U_k \backslash \{u\}\right\}\right\}$;
        if $RC_G(q) = RCmin$ **then** $Nod \leftarrow Nod \backslash V(q)$;
        if $RC_G(q) < RCmin$ **then** $RCmin \leftarrow RC_G(q)$; $Nod \leftarrow V \backslash (V(q) \cup V_0)$;
  else
    for $u = 1, \ldots, M$ **do**
      if $RDV_i^u > RCmin$ **then**
        $V_0 \leftarrow V_0 \cup \{i\}$; **break**;
return V_0;

Because $RC_G(p^{1,s_1}(G)) \geq \min\{RC_G(q) : q \in Q\}$, it should be noticed that all the robust 0-persistent nodes detected by Algorithm 6 are also detected by Algorithm 7, when the cost lower-bound $RCmin$ in the latter algorithm attains a value that is smaller than the minimum path robust deviations that identified robust 0-persistent nodes in the static version. To set such condition, let $RCmin^s$ ($RCmin^d$) represent the cost lower-bound for the static (dynamic) algorithms and V_0^s be the set of robust 0-persistent nodes identified by the static version. Then, the $RCmin^d$ set in some point of Algorithm 7 must satisfy

$$RCmin^d < \min\left\{RDV_i^u : i \in V_0^s \text{ and } u = \min\{u' \in U_M : RDV_i^{u'} > RCmin^s\}\right\}.$$

Under this condition, one may check that all the nodes of V_0^s are scanned and identified as robust 0-persistent by the dynamic strategy as well. In fact, every node $i \in V_0^s$, satisfies

$RDV_i^u = RD_G^{s_u}(p_{1i}^{1,s_u}(G) \diamond p_{in}^{1,s_u}(G)) > RCmin^s$, for some $u \in U_M$. Let q' be a $(1,n)$-path computed for the tests in Algorithm 7, with the robustness cost $RCmin^d$ set above, which satisfies $RCmin^d < RDV_i^u$. Then, $i \notin V(q')$ holds. Otherwise, the definition of robustness cost and the fact that $p_{1i}^{1,s_u}(G) \diamond p_{in}^{1,s_u}(G)$ is the shortest $(1,n)$-path in G containing node i, yield $RCmin^d = RC_G(q') \geq RD_G^{s_u}(p_{1i}^{s_u}(G) \diamond p_{in}^{1,s_u}(G)) = RDV_i^u$, which contradicts the latter condition. Consequently, the nodes of $V \backslash V(q)$ considered in Nod of Algorithm 7, necessarily contain node i. Therefore, this node is scanned and identified as robust 0-persistent by the dynamic version with a cost lower-bound not greater than $RCmin^d$.

The following example illustrates one of the cases which satisfies the condition set above, since both static and dynamic methods start with the same cost lower-bound, i.e. $RC_G(p^{1,s_1}(G)) = \min\{RC_G(q) : q \in Q\}$.

Example Next, Algorithm 7 preprocesses the robust 0-persistent nodes in network G_4 of Figure 3.2. For that, the trees of Figures 3.3 and 3.5 are considered again.

Algorithm 7 starts with

$$RCmin = RC_{G_4}(p^{1,1}(G_4)) = 5 \quad \text{and} \quad Nod = V \backslash V(p^{1,1}(G_4)) = \{3,4,5,6\}.$$

In the following, the algorithm takes into account the maximum number $M \in \{1,2\}$ of scenarios used in the tests (3.3).

- $M = 1$

  Starting by scanning node 3, condition (3.3) is not satisfied for scenario 1. Since $p_{13}^{1,1}(G_4) \diamond p_{37}^{1,1}(G_4) = \langle 1,3,2,7 \rangle$, with $RC_{G_4}(\langle 1,3,2,7 \rangle) = 3$, $RCmin$ is updated to

  $$RCmin = 3.$$

  Additionally, since at this point $V_0 = \emptyset$, Nod is updated to

  $$Nod = V \backslash V(\langle 1,3,2,7 \rangle) = \{4,5,6\}.$$

  For the new $RCmin$, when choosing nodes 4, 5 and 6 to scan, inequality (3.3) is always satisfied for scenario 1, given that

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

  Consequently, all the nodes in Nod are robust 0-persistent, i.e.,

  $$V_0 = \{4,5,6\}.$$

- $M = 2$

  Condition (3.3) does not hold 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, $p_{13}^{1,u}(G_4) \diamond p_{37}^{1,u}(G_4)$, $u \in U_2$, 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

$$V_0 = \{4, 5, 6\}.$$

Computing a robust shortest path after preprocessing As the nodes 4, 5 and 6 were identified as robust 0-persistent, they are removed from G_4, as well as the arcs that start or end at these nodes. Figure 3.7 represents the obtained reduced network.

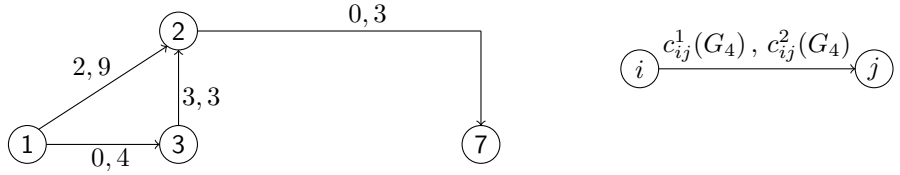


Figure 3.7: Reduced network of G_4 after preprocessing robust 0-persistent nodes with Algorithm 7

Analogously to the network in Figure 3.4, there are only two $(1, 7)$-paths in the reduced network above, $p^{1,1}(G_4) = \langle 1, 2, 7 \rangle$, with $RC_{G_4}(p^{1,1}(G_4)) = 5$, and $q = \langle 1, 3, 2, 7 \rangle$, with $RC_{G_4}(q) = 3$. Again, one concludes that q is the robust shortest path in G_4.

Like for the robust 1-persistency of arcs, in this case the dynamic method was more effective than the static method for the robust 0-persistency of nodes. In fact, with Algorithm 6, only node 5 could be detected as robust 0-persistent, while, with Algorithm 7, besides that node, nodes 4 and 6 were also detected as robust 0-persistent. This allowed to reduce more the network, and, therefore to determine a robust shortest path faster.

For comparison with the static and dynamic approaches, the detection of robust 0-persistent nodes in G_4 is now made through application of Corollary 3.3, assuming that robust 1-persistent arcs have been previously identified.

Application of Corollary 3.3 The arc $(2, 7)$ was identified as robust 1-persistent, when associated to path $q = \langle 1, 3, 2, 7 \rangle$, with $U(2, 7) = \{1\}$. According to Figures 3.3 and 3.5,

$$(2, 7) \notin A(p_{1i}^{1,1}(G_4)) \text{ and } (2, 7) \notin A(p_{i7}^{1,1}(G_4)), \text{ for any } i \in V \backslash V(q) = \{4, 5, 6\}.$$

Therefore, it can be concluded that nodes 4, 5 and 6 are robust 0-persistent.

Computing a robust shortest path after applying Corollary 3.3 The reduced network of G_4 is the same as in Figure 3.7, but now the arc $(2, 7)$ should be represented by a thick line. However, since all the $(1, 7)$-paths in that network contain arc $(2, 7)$, the determination of a robust shortest path is analogous to solving the problem after preprocessing dynamically the robust 0-persistent nodes of G_4.

To conclude, it should be noticed that, for network G_4, Algorithm 7 identifies the same set of nodes as the application of Corollary 3.3, even when $M = 1$. This shows that the dynamic approach may be a good alternative method for preprocessing robust 0-persistent nodes when Corollary 3.3 cannot be applied, that is, when no robust 1-persistent arcs have been detected.

3.4 Computational experiments

This section is dedicated to the empirical comparison between the Algorithms 6 and 7 for preprocessing robust 0-persistent nodes and to their impact on solving the robust shortest path problem with the labeling and the hybrid algorithms (LA and HA) introduced in Chapter 2. 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, $d \in \{1, 2\}$. In fact, in these cases, the majority of the arcs of G belongs to the paths $p^{1,s_u}(G)$, $u \in U_k$, which improves the chances of finding robust 1-persistent arcs.

Algorithms 6 and 7 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 used Dijkstra's algorithm [1] to solve the single source and single destination shortest path problem for a given scenario. As mentioned above, the preprocessing techniques were combined with LA and HA. The robust shortest path problem was solved with and without preprocessing.

The benchmarks used in the experiments correspond to randomly generated directed graphs with $n \in \{500, 1000, 2000, 5000\}$, $d \in \{5, 10, 20\}$ and $k \in \{2, 3\}$. For each scenario, each arc is assigned with a random integer cost in $U(0, 100)$. For each network dimension, 10 instances were generated. For each instance, the two preprocessing algorithms were applied, and condition (3.3) was tested for the scenarios $1, \ldots, M$, with $M \in \{1, \ldots, k\}$. 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.

In order to analyze the performance of Algorithms 6 and 7, 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 average number of robust 0-persistent nodes. The application of the static and the dynamic methods is distinguished by the indices s and d, respectively.

The average number of detected robust 0-persistent nodes and the average CPU times are reported in Tables 3.1, 3.2 and 3.3. In Tables 3.2 and 3.3, 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 3.8 and 3.9 show the average CPU times for $k = 2$ and $k = 3$, respectively, depending on the density of the network.

			$n = 500$		$n = 1000$		$n = 2000$		$n = 5000$	
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 3.1: Number of detected robust 0-persistent nodes

Tables 3.2 and 3.3 and Figures 3.8 and 3.9 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 work demanded by Algorithm 7 being heavier than the required by Algorithm 6, $P_0^s < P_0^d$, the additional effort of the dynamic version leads to the detection of more robust 0-persistent nodes, $N_0^s < N_0^d$ – Table 3.1. 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 3.2: CPU times 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 3.3: CPU times for preprocessing robust 0-persistent nodes, $n \in \{2000, 5000\}$

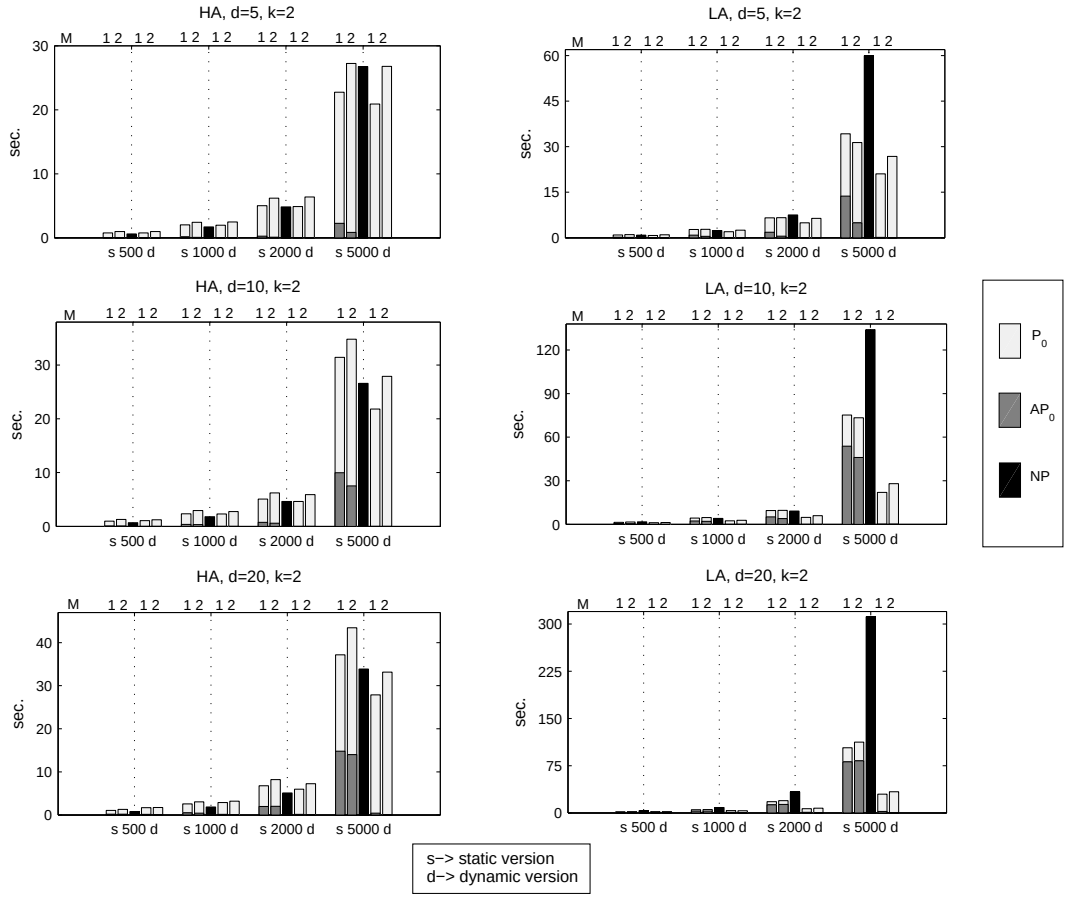


Figure 3.8: CPU times for preprocessing robust 0-persistent nodes and for algorithms HA and LA, with and without preprocessing, when $k = 2$

For each fixed n, d and k, the smaller the number of scenarios for testing (3.3), the less effort is 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 shown in Table 3.1. In general, the values of M that provide the possibility of finding a robust shortest path with preprocessing 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 by not more than $NP - P_0$. Tables 3.2 and 3.3 show that Algorithm 7 was more effective than Algorithm 6 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.

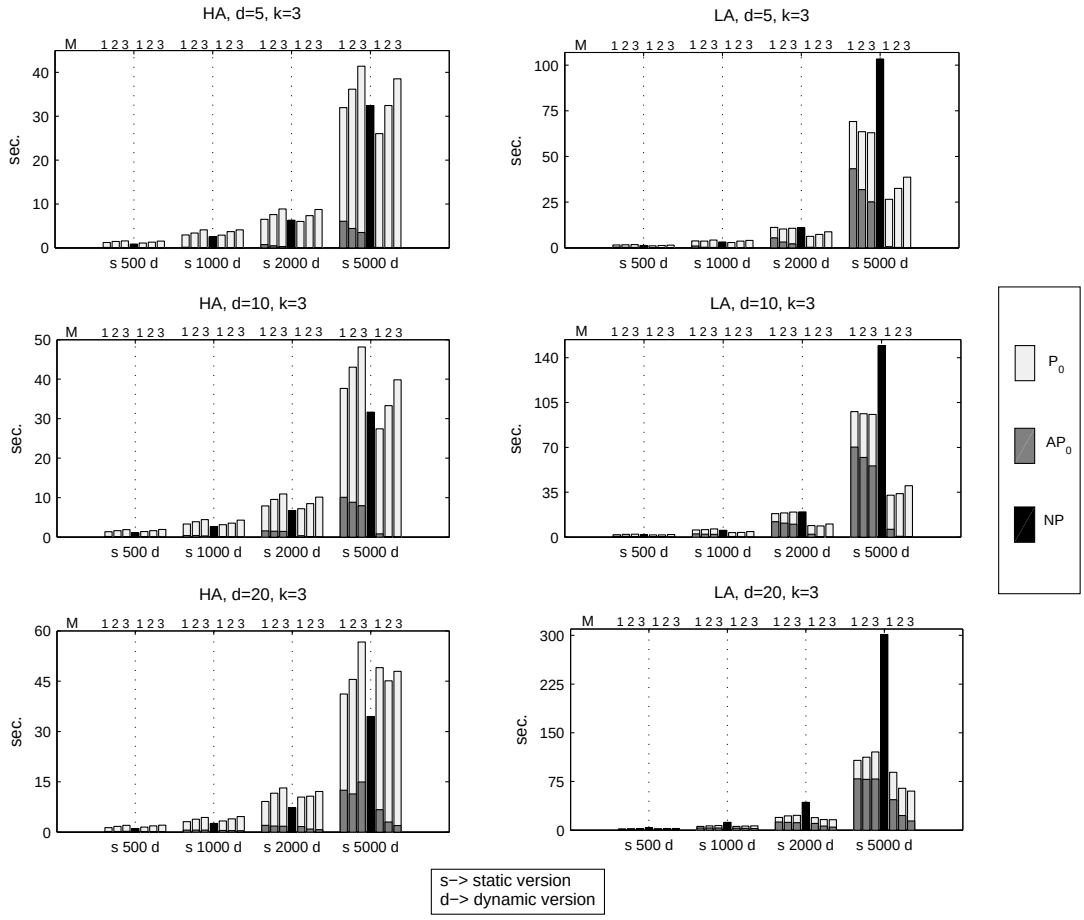


Figure 3.9: CPU times for preprocessing robust 0-persistent nodes and for algorithms HA and LA, with and without preprocessing, when $k = 3$

LA was always more sensitive to preprocessing than HA, and showed the most drastic reductions with respect to NP. This is because removing nodes from the network allows to discard a considerable number of labels in LA, making easier the search for 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, as the search for a robust shortest path is focuses on selecting suitable deviation arcs and this can be done in few iterations without preprocessing.

The number of detected robust 0-persistent nodes is high for the networks with the lowest densities ($d \in \{5, 10\}$), particularly for Algorithm 7 – Table 3.1. 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 3.8 and 3.9 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, $|NP - TP_0^s| < |NP - TP_0^d|$.

3.5 Conclusions

In this chapter, new techniques were developed to identify robust 1-persistent arcs and robust 0-persistent nodes in G. The former are contained in all robust shortest paths and can be found among the arcs of $p^{1,s_u}(G)$, $u \in U_k$. The latter are not part of any optimal solution and can be any node, except 1 and n. The presented methods followed static [40] and dynamic [41] approaches for preprocessing. Both are based on comparisons with established lower-bounds, which are fixed along the algorithm in the static version and which are updated, according to the least robustness cost of the computed $(1, n)$-paths, in the dynamic version. The set of arcs preprocessed by the dynamic strategy contain the set of arcs identified by the static version. However, for the nodes, the same happens if the minimum cost lower-bound in the dynamic method becomes smaller than the minimum path robust deviations that identified robust 0-persistent nodes in the static version. The pseudo-codes of the algorithms were presented and they were shown to have polynomial time complexities.

Since the robust 1-persistent arcs are restricted to the arcs of $p^{1,s_u}(G)$, $u \in U_k$, the chances of detecting them reduced drastically when dense networks were considered. Nevertheless, when some robust 1-persistent arc can be identified, the search for robust 0-persistent nodes becomes less demanding in terms of computational effort, as the calculation of the path robust deviations can be avoided. For the identification of robust 0-persistent nodes, the number of scenarios used in the tests was limited as well. The preprocessing approaches were exemplified. The performances of the dynamic and the static procedures were empirically tested on random instances, when combined with the labeling and hybrid algorithms introduced in Chapter 2.

For preprocessing robust 0-persistent nodes, the performed experiments revealed that, in general, the dynamic approach is the best choice. Besides, LA was always more efficient after preprocessing than with no preprocessing at all. The same happened with HA using the dynamic processing for networks with a large number of nodes, even for the cases for which the static approach was not efficient, and using only the first scenario in the tests. 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 20675%, when $N_0^s \neq 0$. In general, this reduction was also more demanding in terms of the CPU times. Nevertheless, in 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 overall 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.

Chapter 4

Reoptimization methods for the robust shortest path problem

Reoptimization techniques may be particularly useful when a sequence of closely related instances of the same problem has to be solved, or simply when the network conditions change after the optimal solution is known. The goal of such techniques is to reduce the cost of solving a new problem from scratch, by using information about the optimal solution of the preceding problem. The present chapter is dedicated to develop reoptimization methods for the robust shortest path problem in case the set of arcs, or the set of scenarios, changes due to the deletion or the inclusion of some elements. The first question that is raised, for any case, is whether the optimal solution of the original problem is still optimal in the modified network. Conditions are established in order to check if this is the case. Otherwise, reoptimization methods can be applied. Therefore, different methods are developed, first addressing changes in the set of scenarios, and later addressing changes in the set of arcs. It is assumed that an optimal solution, as well as some other related parameters, are known. The idea behind the introduced approaches is to apply a labeling technique combined with pruning rules, which aims at constructing a tree of paths starting from the previous information. For each of the studied cases, the algorithms are outlined and their complexity in terms of operations is evaluated. Finally, the methods are exemplified for small instances.

4.1 Introduction

In this chapter, reoptimization methods for the robust shortest path problem will be designed. The reoptimization of several classic optimization problems has been the subject of research in the literature. Such problems include both those that can be solved in polynomial time as well as NP-hard problems. For instance, shortest path problems [17, 24, 36, 37], the minimum

spanning tree problem [15], the minimum cost flow problem [37], the knapsack problem [2], or the traveling salesman problem [4, 9], to name a few. Reoptimization algorithms for combinatorial optimization in general were addressed in [3, 47]. The insertion or deletion of nodes or arcs in the network, or the modification of the costs are the typical changes considered in those works.

The modifications addressed in the following concern including, and deleting, scenarios in, and from, the network, and afterwards including, and deleting arcs. A simple robust shortest path in G, $\tilde{p}$, is supposed to be known, as well as its robustness cost and the shortest $(1,n)$-path costs for all scenarios. This assumption is not too strong, given that these latter values are required for computing the robustness costs. In order to ensure that there exists an optimal solution that is simple after the network modifications, it is considered that the new networks still do not contain negative cost cycles for any scenario. The preceding chapter included the identification of arcs (nodes) that are part (not part) of any optimal solution. Naturally, changes that involve these arcs or nodes do not affect the robust shortest path. However, for the general case, conditions are established which ensure that $\tilde{p}$ remains an optimal solution after modifying the network G. If this is not so, then the next step is to reoptimize $\tilde{p}$, and, therefore, methods to find a new optimal solution are developed. With that aim, a set of $(1,n)$-paths in the new network, which contains the new solution, is defined. Then, the search for that path is performed by constructing a tree of those paths. The search tree, denoted ST, is initialized according with the known information about the previous solution. Afterwards, its construction follows a labeling approach using cost lower-bounds to filter new labels, depending on each reoptimization case. When a $(1,n)$-path is computed, the rules derived for the search set are applied.

Like before, UB will denote an upper-bound for the least robustness cost and *sol* a robust shortest path candidate. List X is used to store the $(1,i)$-paths computed while constructing ST, $i \in V$, and it is managed like a FIFO list.

4.2 Variation of the number of scenarios

This section is devoted to reoptimize the robust shortest path when scenarios are deleted from, or inserted in G. To begin, it is intended to derive the conditions under which $\tilde{p}$ is maintained as the robust shortest path in the obtained reduced or extended versions of G. With this purpose, a preliminary general result is established for two networks, such that one results from the other by deleting or inserting scenarios. Under these conditions, both networks have the same paths, being presented the relation between the robust deviations for the common scenarios and between the robustness costs of the $(1,n)$-paths in the two networks.

Lemma 4.1. *Let $G'_S = G'_S(V, A, S')$ and $G''_S = G''_S(V, A, S'')$ be networks, such that $S' = \{s'_u : u \in U'\}$ and $S'' = S' \cup \{s''_u : u \in U^*\}$ are both finite. Then, for any $p \in P_{1n}(G'_S) = P_{1n}(G''_S)$,*

1. *$RD^{s'_u}_{G'_S}(p) = RD^{s'_u}_{G''_S}(p)$, for any $u \in U'$.*
2. *$RC_{G''_S}(p) = \max\left\{RC_{G'_S}(p)\,,\, \max_{u\in U^*} RD^{s''_u}_{G''_S}(p)\right\}$.*
3. *$RC_{G'_S}(p) \leq RC_{G''_S}(p)$.*

Proof.

1. Let s'_u, $u \in U'$, be any scenario common to G'_S and G''_S. By assumption, G'_S and G''_S have the same paths, therefore,
$$LB^{s'_u}_{1n}(G'_S) = LB^{s'_u}_{1n}(G''_S).$$
Moreover, for any $p \in P_{1n}(G'_S) = P_{1n}(G''_S)$,
$$c^{s'_u}_{G'_S}(p) = c^{s'_u}_{G''_S}(p)$$
and, consequently,
$$RD^{s'_u}_{G'_S}(p) = RD^{s'_u}_{G''_S}(p).$$
2. Let $p \in P_{1n}(G'_S) = P_{1n}(G''_S)$. By definition of robustness cost in G''_S,
$$RC_{G''_S}(p) = \max\left\{\max_{u\in U'} RD^{s'_u}_{G''_S}(p), \max_{u\in U^*} RD^{s''_u}_{G''_S}(p)\right\}.$$
By 1., one has
$$\max_{u\in U'} RD^{s'_u}_{G''_S}(p) = \max_{u\in U'} RD^{s'_u}_{G'_S}(p) = RC_{G'_S}(p),$$
where the last equality comes from the definition of robustness cost in G'_S. Then, one may write,
$$RC_{G''_S}(p) = \max\left\{RC_{G'_S}(p)\,,\, \max_{u\in U^*} RD^{s''_u}_{G''_S}(p)\right\}.$$
3. The proof is immediate from 2.

□

Lemma 4.1 will be useful when proving results related with the determination of the optimal solution when S is reduced or extended. In either case, the conditions under which $\tilde{p}$ is the robust shortest path in the modified version of G are deduced. From this analysis, the possible candidates, besides $\tilde{p}$, are restricted to specific subsets of $(1, n)$-paths. With this knowledge, the reoptimization methods to search for the new robust shortest path are developed by means of a ST. This will lead to the establishment of three extension rules, the first two, common

to every method and concerned with the extension of paths to any node, but n, and, the third rule, covering the extensions to node n. This rule is adapted according with the search sets obtained. The structure of the algorithmic procedures are outlined, with the deduction of their computational time complexities. The examples for illustrating the application of the algorithms are also provided, in order to show all their possible steps. The removal of scenarios from G is addressed in the following.

4.2.1 Elimination of scenarios

Assume that G is a network with $k > 2$, and let $S^* = \{s_u : u \in U^*\} \subseteq S$ be the set of scenarios deleted from S, with $U^* \subseteq U_k$ and $1 \leq |U^*| = k^* < k-2$. The reduced version of G, $G^-_{S^*}$, has set of scenarios $S \backslash S^* = \{s_u : u \in U_k \backslash U^*\}$ and set of $(1,n)$-paths $P_{1n}(G^-_{S^*}) = P_{1n}(G)$. Corollary 4.2 is a consequence of Lemma 4.1, by considering the networks $G'_S = G^-_{S^*}$ and $G''_S = G$.

Corollary 4.2. *For any $p \in P_{1n}(G^-_{S^*}) = P_{1n}(G)$,*

1. $RD^{s_u}_{G^-_{S^*}}(p) = RD^{s_u}_G(p)$, *for any $u \in U_k \backslash U^*$.*
2. $RC_G(p) = \max\left\{RC_{G^-_{S^*}}(p)\,,\, \max_{u \in U^*} RD^{s_u}_G(p)\right\}$.
3. $RC_{G^-_{S^*}}(p) \leq RC_G(p)$.

According to this result, none of the robustness costs of the paths in $P_{1n}(G)$ increases in $G^-_{S^*}$. The $(1,n)$-paths with a robustness cost in $G^-_{S^*}$ smaller than $RC_{G^-_{S^*}}(\tilde{p})$ belong to a particular subset of $P_{1n}(G^-_{S^*})$. This result is established in the following, and it provides a necessary and sufficient condition for $\tilde{p}$ being a robust shortest path in G and $G^-_{S^*}$.

Proposition 4.3. *Let $\widehat{P}_{1n}(G^-_{S^*}) \subseteq P_{1n}(G^-_{S^*})$ be given by*

$$\widehat{P}_{1n}(G^-_{S^*}) = \{p \in P_{1n}(G^-_{S^*}) : RD^{s_u}_G(p) \geq RC_G(\tilde{p}),\ \textit{for some } u \in U^*\}.$$

1. *If $p \in P_{1n}(G^-_{S^*})$ satisfies $RC_{G^-_{S^*}}(p) < RC_{G^-_{S^*}}(\tilde{p})$, then $p \in \widehat{P}_{1n}(G^-_{S^*})$.*
2. *$\tilde{p}$ is a robust shortest path in $G^-_{S^*}$ if and only if $RC_{G^-_{S^*}}(p) \geq RC_{G^-_{S^*}}(\tilde{p})$, for any $p \in \widehat{P}_{1n}(G^-_{S^*})$.*

Proof.

1. Let $p \in P_{1n}(G^-_{S^*})$ satisfy $RC_{G^-_{S^*}}(p) < RC_{G^-_{S^*}}(\tilde{p})$. Then, applying point 3. of Corollary 4.2 to $\tilde{p}$ yields
$$RC_{G^-_{S^*}}(p) < RC_G(\tilde{p}).$$

Because, by assumption, $\tilde{p}$ is a robust shortest path in G,

$$RC_G(\tilde{p}) \leq RC_G(p), \tag{4.1}$$

and, consequently,

$$RC_{G^-_{S^*}}(p) < RC_G(p).$$

Then, from 2. of Corollary 4.2, it follows that

$$RC_G(p) = \max_{u \in U^*} RD^{s_u}_G(p).$$

Because of (4.1), one has

$$\max_{u \in U^*} RD^{s_u}_G(p) \geq RC_G(\tilde{p}),$$

or, equivalently,

$$RD^{s_u}_G(p) \geq RC_G(\tilde{p}), \text{ for some } u \in U^*,$$

i.e., $p \in \widehat{P}_{1n}(G^-_{S^*})$.

2. Let $\tilde{p}$ be a robust shortest path in $G^-_{S^*}$. Then, $RC_{G^-_{S^*}}(p) \geq RC_{G^-_{S^*}}(\tilde{p})$, for any $p \in P_{1n}(G^-_{S^*})$, and, in particular, when $p \in \widehat{P}_{1n}(G^-_{S^*})$.

   Assume now that $RC_{G^-_{S^*}}(p) \geq RC_{G^-_{S^*}}(\tilde{p})$ holds, for any $p \in \widehat{P}_{1n}(G^-_{S^*})$. This condition occurs when $p \in P_{1n}(G^-_{S^*}) \backslash \widehat{P}_{1n}(G^-_{S^*})$, according with 1., and, therefore, it is satisfied for any $p \in P_{1n}(G^-_{S^*})$. Consequently, $\tilde{p}$ is a robust shortest path in $G^-_{S^*}$.

□

When $RC_{G^-_{S^*}}(\tilde{p}) \neq 0$, the result above allows to restrict the search for a robust shortest path in $G^-_{S^*}$ to the simple paths in $\widehat{P}_{1n}(G^-_{S^*})$ with the least robustness cost in $G^-_{S^*}$.

In the following, the reoptimization procedure for $G^-_{S^*}$ is devised, according with the search strategy described in the beginning of the chapter. The algorithm sets path $\tilde{p}$ as the first candidate *sol* for the optimal solution and $RC_{G^-_{S^*}}(\tilde{p})$ as the first upper-bound UB for the optimal value. If $UB = 0$, $\tilde{p}$ is returned as the optimal solution of $G^-_{S^*}$, else ST has to be constructed. Next, the extension rules for the paths in ST are explained for $G^-_{S^*}$.

The method is supported by assigning labels to each path in ST and those are generated by the same rules used for the labeling algorithm in Chapter 2. Since k^* is the number of scenarios removed from G, the label in $G^-_{S^*}$ for each $p_{1i} \in ST \cap P_{1i}(G^-_{S^*})$, $i \in V$, is a $(k - k^*)$-vector given by

$$z_{G^-_{S^*}}(p_{1i}) = (z^1_{G^-_{S^*}}(p_{1i}), \ldots, z^{k-k^*}_{G^-_{S^*}}(p_{1i})).$$

It should be noted that the position of each component in the label does not necessarily coincide with the position of the associate scenario in $S\backslash S^*$, or, equivalently, with the position of the scenario index in $U_k\backslash U^*$. Therefore, in order to set each label, the relation between the previous positions must be known, by means of the following function

$$\begin{array}{rccc} \rho: & U_{k-k^*} & \longmapsto & U_k\backslash U^* \\ & u & \longmapsto & u\text{-th element in } U_k\backslash U^* \end{array}$$

Under these conditions, the first label is set for node 1 as

$$z_{G^-_{S^*}}(\langle 1\rangle) = (-LB_{1n}^{s_{\rho(1)}}(G), \ldots, -LB_{1n}^{s_{\rho(k-k^*)}}(G)),$$

since $LB_{1n}^{s_{\rho(u)}}(G^-_{S^*}) = LB_{1n}^{s_{\rho(u)}}(G)$, for any $u \in U_{k-k^*}$.

Let $p_{1i} \in ST \cap P_{1i}(G^-_{S^*})$, $i \in V$, for which $(i,j) \in A$ is added, then, the label in $G^-_{S^*}$ for the $(1,j)$-path $p_{1j} = p_{1i} \diamond \langle i,j\rangle$, is obtained from $z_{G^-_{S^*}}(p_{1i})$, as follows

$$z_{G^-_{S^*}}(p_{1j}) = (z^1_{G^-_{S^*}}(p_{1i}) + c_{ij}^{s_{\rho(1)}}(G^-_{S^*}), \ldots, z^{k-k^*}_{G^-_{S^*}}(p_{1i}) + c_{ij}^{s_{\rho(k-k^*)}}(G^-_{S^*})).$$

When $j = n$,

$$z_{G^-_{S^*}}(p_{1n}) = (RD_{G^-_{S^*}}^{s_{\rho(1)}}(p_{1n}), \ldots, RD_{G^-_{S^*}}^{s_{\rho(k-k^*)}}(p_{1n})),$$

and, consequently, the robustness cost of p_{1n} in $G^-_{S^*}$ is

$$RC_{G^-_{S^*}}(p_{1n}) = \max_{u\in U_{k-k^*}} z^u_{G^-_{S^*}}(p_{1n}).$$

The first extension rule consists in considering in ST the paths of $P_{1i}(G^-_{S^*})$, $i \in V\backslash\{n\}$, that can be extended from node i to $(1,n)$-paths with robustness costs in $G^-_{S^*}$ that can improve UB, i.e. such that

$$\max_{u\in U_{k-k^*}} \{z^u_{G^-_{S^*}}(p_{1i}) + LB_{in}^{s_{\rho(u)}}(G^-_{S^*})\} < UB. \tag{4.2}$$

By the same reason, the second extension rule considers that an arc (i,j) can be added to a path $p_{1i} \in ST \cap P_{1i}(G^-_{S^*})$, $i \in V\backslash\{n\}$, when

$$\max_{u\in U_{k-k^*}} \{z^u_{G^-_{S^*}}(p_{1i}) + c_{ij}^{s_{\rho(u)}}(G^-_{S^*}) + LB_{jn}^{s_{\rho(u)}}(G^-_{S^*})\} < UB. \tag{4.3}$$

From node 1, ST starts to include the arcs of $\tilde{p}$ at a time, since its first until its penultimate, that satisfy the rule above. Then, list X, which collects the paths for extension in ST, starts with

$$X = \{\langle 1\rangle\} \cup \{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in A(\tilde{p}),\ j \neq n,\ \text{satisfy (4.3)}\}.$$

One stops to include the arcs of $\tilde{p}$ in ST, when an arc that does not satisfy (4.3) is found. Once the part of path $\tilde{p}$ is set in ST, the subsequent extension must not repeat the sub-paths

of $\tilde{p}$ in ST and, besides, it must not produce loops, since it is intended to find a simple robust shortest path in $G^-_{S^*}$. With the first goal, any arc (i,j) added to the $(1,i)$-sub-path of $\tilde{p}$, $\tilde{p}_{1i}$, $i \in V\backslash\{n\}$, in ST cannot be part of $\tilde{p}$. With the second goal, the same extension applied to a simple path $p_{1i} \in ST \cap P_{1i}(G^-_{S^*})$ must assure that p_{1j} is still simple. Hence, by knowing the part of $\tilde{p}$ in ST, the set of arcs candidate to extend $p_{1i} \in ST \cap P_{1i}(G^-_{S^*})$, $i \in V\backslash\{n\}$, is denoted by $Ad_{G^-_{S^*}}(p_{1i}\,|\,\tilde{p})$, and it can be determined as

$$Ad_{G^-_{S^*}}(p_{1i}\,|\,\tilde{p}) = \begin{cases} \Big\{(i,j) \in A\backslash A(\tilde{p}) : j \notin V(p_{1i})\Big\} & \text{if} \quad p_{1i} = \tilde{p}_{1i} \\ \\ \Big\{(i,j) \in A : j \notin V(p_{1i})\Big\} & \text{if} \quad p_{1i} \neq \tilde{p}_{1i} \end{cases}$$

If $(i,j) \in Ad_{G^-_{S^*}}(p_{1i}\,|\,\tilde{p})$, with $j \neq n$, then (i,j) is added to p_{1i}, when (4.3) is satisfied. If $(i,n) \in Ad_{G^-_{S^*}}(p_{1i}\,|\,\tilde{p})$, a third extension rule concerning the arcs with head node n must apply. Specifically, (i,n) extends p_{1i} if it produces a $(1,n)$-path p_{1n} belonging to $\widehat{P}_{1n}(G^-_{S^*})$, according with Proposition 4.3, and if $RC_{G^-_{S^*}}(p_{1n})$, denoted by $RCaux$, can improve UB, i.e. if

$$\begin{cases} RD^{s_u}_G(p_{1n}) \geq RC_G(\tilde{p}), \text{ for some } u \in U^* \\ \\ RCaux = \max_{u \in U_{k-k^*}}\{z^u_{G^-_{S^*}}(p_{1i}) + c^{s_{\rho(u)}}_{in}(G^-_{S^*})\} < UB. \end{cases} \tag{4.4}$$

When both conditions are satisfied, the new candidate for the robust shortest path of $G^-_{S^*}$, sol, is p_{1n}, and UB is updated to $RCaux$. Along the method, whenever UB is improved, all the paths collected in X must be tested again in order to check if some can be discarded. This happens when the path does not satisfy (4.2) for the new update.

The pseudo-code of the reoptimization method is given in Algorithm 8.

Computational time complexity order Let $\underline{k} = k - k^*$ denote the number of scenarios in $G^-_{S^*}$. Algorithm 8 is performed in two stages. The first consists in determining the trees $\mathcal{T}^{s_u}_n(G^-_{S^*})$, the costs $LB^{s_u}_{in}(G^-_{S^*})$, $i \in V$, $u \in U_k\backslash U^*$, and setting the initial UB. The former task can be done in $O^a_1 = \mathcal{O}(\underline{k}m)$ time for acyclic networks and in $O^c_1 = \mathcal{O}(\underline{k}(m + n\log n))$ time for general networks, as seen before. These are the complexities for the first stage, because they are not affected by the complexity for calculating $RC_{G^-_{S^*}}(\tilde{p})$, in order to initialize UB, which is $\mathcal{O}(\underline{k}n)$.

Algorithm 8: Finding the robust shortest path in $G^-_{S^*}$, given $\tilde{p}$, $RC_G(\tilde{p})$ and $LB^{s_u}_{1n}(G)$, $u \in U_k$

for $u \in U_k \backslash U^*$ **do** Compute $\mathcal{T}^{s_u}_n(G^-_{S^*})$ and $LB^{s_u}_{in}(G^-_{S^*})$, $i \in V$;
$RC_{G^-_{S^*}}(\tilde{p}) \leftarrow \max_{u \in U_k \backslash U^*} RD^{s_u}_{G^-_{S^*}}(\tilde{p})$; $UB \leftarrow RC_{G^-_{S^*}}(\tilde{p})$; $sol \leftarrow \tilde{p}$;
if $UB \neq 0$ **then**
  $X \leftarrow \{\langle 1 \rangle\}$;
  for $u \in U_{k-k^*}$ **do** $\rho(u) \leftarrow u$-th element in $U_k \backslash U^*$; $z^u_{G^-_{S^*}}(\langle 1 \rangle) \leftarrow -LB^{s_{\rho(u)}}_{1n}(G^-_{S^*})$;
  for $(i,j) \in A(\tilde{p})$ ***and*** $j \neq n$ **do**
    if $\max_{u \in U_{k-k^*}} \{z^u_{G^-_{S^*}}(\tilde{p}_{1i}) + c^{s_{\rho(u)}}_{ij}(G^-_{S^*}) + LB^{s_{\rho(u)}}_{jn}(G^-_{S^*})\} < UB$ **then**
      $X \leftarrow X \cup \{\tilde{p}_{1j}\}$;
      for $u \in U_{k-k^*}$ **do** $z^u_{G^-_{S^*}}(\tilde{p}_{1j}) \leftarrow z^u_{G^-_{S^*}}(\tilde{p}_{1i}) + c^{s_{\rho(u)}}_{ij}(G^-_{S^*})$;
    else break;
while $X \neq \emptyset$ **do**
  $p_{1i} \leftarrow$ first path in X; $X \leftarrow X - \{p_{1i}\}$; Compute $Ad_{G^-_{S^*}}(p_{1i} \,|\, \tilde{p})$;
  for $(i,j) \in Ad_{G^-_{S^*}}(p_{1i} \,|\, \tilde{p})$ **do**
    $p_{1j} \leftarrow p_{1i} \diamond \langle i,j \rangle$;
    if $j = n$ **then**
      for $u \in U^*$ **do**
        if $RD^{s_u}_G(p_{1j}) \geq RC_G(\tilde{p})$ **then**
          $RCaux \leftarrow \max_{u' \in U_{k-k^*}} \{z^{u'}_{G^-_{S^*}}(p_{1i}) + c^{s_{\rho(u')}}_{ij}(G^-_{S^*})\}$;
          if $RCaux < UB$ **then**
            $UB \leftarrow RCaux$; $sol \leftarrow p_{1j}$;
            for $p_{1i'} \in X$ **do**
              for $u' \in U_{k-k^*}$ **do**
                if $z^{u'}_{G^-_{S^*}}(p_{1i'}) + LB^{s_{\rho(u')}}_{i'n}(G^-_{S^*}) \geq UB$ **then**
                  $X \leftarrow X - \{p_{1i'}\}$; **break**;
          break;
    else
      if $\max_{u \in U_{k-k^*}} \{z^u_{G^-_{S^*}}(p_{1i}) + c^{s_{\rho(u)}}_{ij}(G^-_{S^*}) + LB^{s_{\rho(u)}}_{jn}(G^-_{S^*})\} < UB$ **then**
        $X \leftarrow X \cup \{p_{1j}\}$;
        for $u \in U_{k-k^*}$ **do** $z^u_{G^-_{S^*}}(p_{1j}) \leftarrow z^u_{G^-_{S^*}}(p_{1i}) + c^{s_{\rho(u)}}_{ij}(G^-_{S^*})$;
return sol;

The second stage concerns the construction of ST, which requires the generation of labels and the application of the extension rules. In order to set the labels, function ρ is defined in $\mathcal{O}(\underline{k})$ time, which is also the complexity for generating each label for the sub-paths of $\tilde{p}$, in order to set the initial list X. Similarly to the labeling algorithm in Chapter 2, the maximum number of paths generated in $ST \cap P_{1i}(G^-_{S^*})$, $i \in V$, denoted by W^-_S, is considered. Then, $(n-1)W^-_S$ is the maximum number of paths in X, that is the maximum number of iterations of the **while** loop in line 11, and each of them requires at most $n-1$ iterations of the **for** loop in line 13 for scanning the arcs for addition. In each of these iterations, the remaining labels are generated and the extension rules are applied. The labeling and the extension rule (4.3) are performed in $\mathcal{O}(\underline{k})$ time. For extension rule (4.4), $\mathcal{O}(k^*n)$ operations are demanded to test the first inequality in a worst case, and $\mathcal{O}(\underline{k})$ operations are required to set $RCaux$. The extension rule (4.2) is applied in case UB is updated and requires $\mathcal{O}(\underline{k})$ time to test each path in X, and, therefore, performing the **for** loop in line 21 has $\mathcal{O}(\underline{k}nW^-_S)$ time. Consequently, the second stage has a complexity of $O_2 = \mathcal{O}(k^*n^3W^-_S + \underline{k}n^3(W^-_S)^2) = \mathcal{O}(\max\{k^*, \underline{k}W^-_S\}n^3W^-_S)$.

In conclusion, Algorithm 8 has a total time complexity of $\mathcal{O}\big(\max\{k^*, \underline{k}W^-_S\}n^3W^-_S\big)$, for any type of network, since $\log n \ll n$ and $m < n^2$. It can be noted that besides the number of scenarios of $G^-_{S^*}$, the time complexity depends also on the number of scenarios removed from G.

Example In the following, Algorithm 8 is applied after reducing the network $G_5 = G_5(V, A, U_4)$ in Figure 4.1, with respect to its set of scenarios.

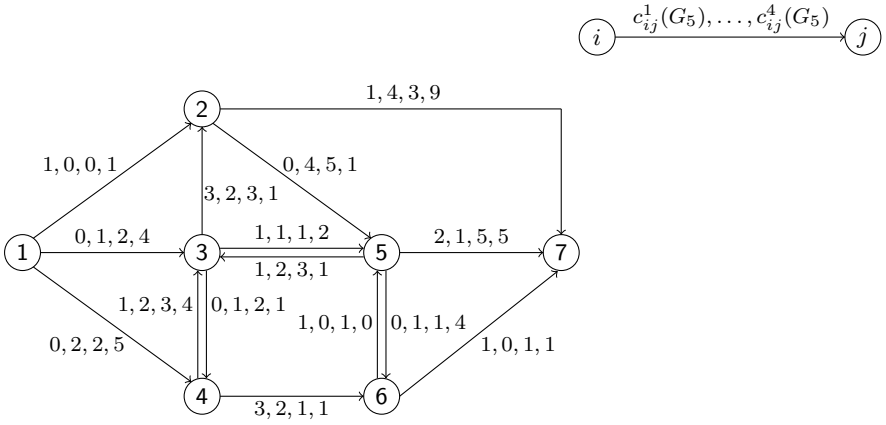


Figure 4.1: Network G_5

By hypothesis, the shortest path costs

$$LB^1_{17}(G_5) = 2\,,\; LB^2_{17}(G_5) = 3\,,\; LB^3_{17}(G_5) = 3 \text{ and } LB^4_{17}(G_5) = 6$$

are known, as well as a simple robust shortest path of G_5,

$$\tilde{p} = \langle 1, 4, 6, 7 \rangle\,, \text{ with } RC_{G_5}(\tilde{p}) = 2.$$

The reduced network, $(G_5)^-_{\{2,3\}}$, results from G_5 by removing scenarios 2 and 3 – Figure 4.2. In this example, $\mathcal{T}_7^u((G_5)^-_{\{2,3\}}) = \mathcal{T}_7^u(G_5)$, $u \in \{1, 4\}$ – Figure 4.3.

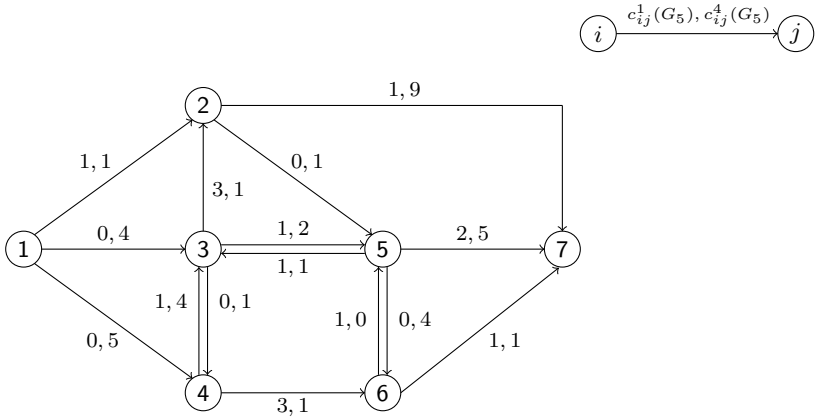
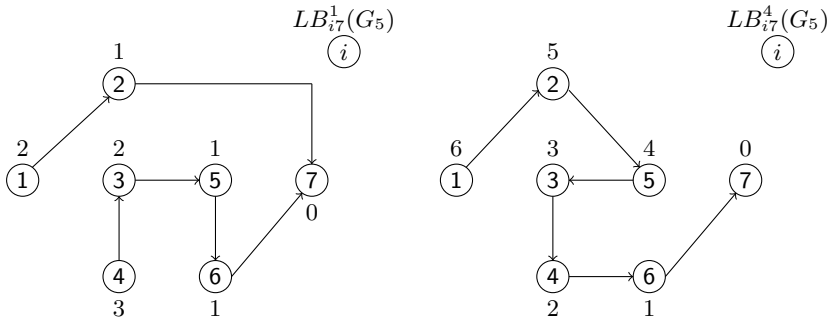


Figure 4.2: Network $(G_5)^-_{\{2,3\}}$



(a) $\mathcal{T}_7^1((G_5)^-_{\{2,3\}}) = \mathcal{T}_7^1(G_5)$ (b) $\mathcal{T}_7^4((G_5)^-_{\{2,3\}}) = \mathcal{T}_7^4(G_5)$

Figure 4.3: Shortest path trees rooted at node 7 in G_5 and $(G_5)^-_{\{2,3\}}$ for scenarios 1 and 4

The initial UB and sol are set to

$$UB = RC_{(G_5)^-_{\{2,3\}}}(\tilde{p}) = \max_{u \in \{1,4\}} RD^u_{(G_5)^-_{\{2,3\}}}(\tilde{p}) = 2 \quad \text{and} \quad sol = \tilde{p} = \langle 1, 4, 6, 7 \rangle.$$

Since $UB \neq 0$, ST has to be constructed. To set the labels, the function $\rho : U_2 \longmapsto \{1, 4\}$ satisfies $\rho(1) = 1$ and $\rho(2) = 4$. Then, the first label is set to

$$z_{(G_5)^-_{\{2,3\}}}(\langle 1 \rangle) = (-LB^1_{17}(G_5), -LB^4_{17}(G_5)) = (-2, -6),$$

and ST starts with the sub-paths of $\tilde{p}$ in list X given by

$$X = \big\{\langle 1\rangle\big\} \cup \big\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in \{(1,4),(4,6)\} \text{ satisfy (4.3)}\big\}.$$

The arc $(1,4)$ can extend $\langle 1\rangle$, because

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1\rangle) + c^{\rho(u)}_{14}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{47}((G_5)^-_{\{2,3\}})\} = 1 < UB.$$

Then, the label

$$z_{(G_5)^-_{\{2,3\}}}(\langle 1,4\rangle) = (-2,-1),$$

is set, and, one concludes that path $\langle 1,4,6\rangle$ does not belong to ST, because

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,4\rangle) + c^{\rho(u)}_{46}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{67}((G_5)^-_{\{2,3\}})\} = 2 \geq UB.$$

Therefore, ST starts with

$$X = \{\langle 1\rangle, \langle 1,4\rangle\}.$$

Figure 4.4 shows the ST obtained when applying Algorithm 8 to $(G_5)^-_{\{2,3\}}$. The part of $\tilde{p}$ which is not included is represented by a dashed line. Moreover, the label in $(G_5)^-_{\{2,3\}}$ for each $(1,i)$-path in ST is attached to node i, $i \in V$.

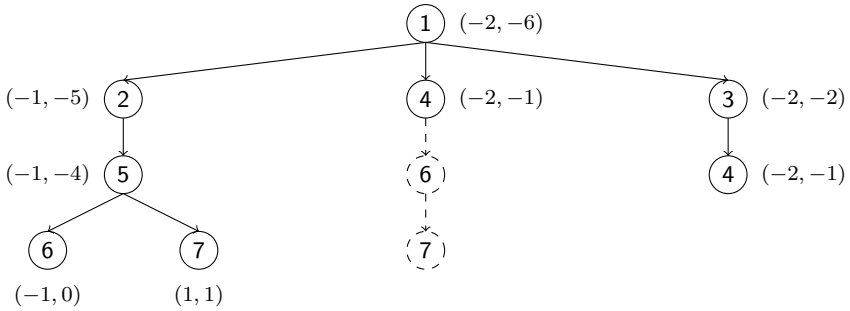


Figure 4.4: ST produced by Algorithm 8 for $(G_5)^-_{\{2,3\}}$

Starting to extend from node 1, one has

$$Ad_{(G_5)^-_{\{2,3\}}}(\langle 1\rangle \,|\, \tilde{p}) = \{(1,j) \in A \backslash A(\tilde{p})\} = \{(1,2),(1,3)\}.$$

All of the arcs in the previous set can be added to node 1, since

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1\rangle) + c^{\rho(u)}_{12}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{27}((G_5)^-_{\{2,3\}})\} = 0 < UB$$

and

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1\rangle) + c^{\rho(u)}_{13}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{37}((G_5)^-_{\{2,3\}})\} = 1 < UB.$$

Consequently, X is updated to

$$X = \{\langle 1,4\rangle, \langle 1,2\rangle, \langle 1,3\rangle\}$$

and the labels

$$z_{(G_5)^-_{\{2,3\}}}(\langle 1,2\rangle) = (-1,-5) \quad \text{and} \quad z_{(G_5)^-_{\{2,3\}}}(\langle 1,3\rangle) = (-2,-2)$$

are set. In the next iteration, path $\langle 1,4\rangle$ is selected for extension. In this case, the arc in

$$Ad_{(G_5)^-_{\{2,3\}}}(\langle 1,4\rangle \,|\, \tilde{p}) = \{(4,j) \in A \backslash A(\tilde{p}) : j \neq 1\} = \{(4,3)\}$$

cannot be added, given that

$$\max_{u \in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,4\rangle) + c^{\rho(u)}_{43}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{37}((G_5)^-_{\{2,3\}})\} = 6 \geq UB.$$

Path $\langle 1,2\rangle$ is the next to be extended in X, with

$$Ad_{(G_5)^-_{\{2,3\}}}(\langle 1,2\rangle \,|\, \tilde{p}) = \{(2,j) \in A : j \neq 1\} = \{(2,5),(2,7)\}.$$

Arc $(2,5)$ is added, because

$$\max_{u \in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,2\rangle) + c^{\rho(u)}_{25}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{57}((G_5)^-_{\{2,3\}})\} = 0 < UB.$$

The label

$$z_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5\rangle) = (-1,-4),$$

is then obtained and X is updated to

$$X = \{\langle 1,3\rangle, \langle 1,2,5\rangle\}.$$

Nevertheless, when considering arc $(2,7)$ and the scenarios of $S^* = \{2,3\}$, one has

$$RD^2_{G_5}(\langle 1,2,7\rangle) = 1 < RC_{G_5}(\tilde{p}) \quad \text{and} \quad RD^3_{G_5}(\langle 1,2,7\rangle) = 0 < RC_{G_5}(\tilde{p}).$$

This means that the first condition of (4.4) is not satisfied. Consequently, $\langle 1,2,7\rangle$ cannot be an optimal solution in $(G_5)^-_{\{2,3\}}$ and, therefore, arc $(2,7)$ is not added to $\langle 1,2\rangle$. At the next iteration, $\langle 1,3\rangle$ is evaluated, with

$$Ad_{(G_5)^-_{\{2,3\}}}(\langle 1,3\rangle \,|\, \tilde{p}) = \{(3,j) \in A : j \neq 1\} = \{(3,2),(3,4),(3,5)\}.$$

The arcs $(3,2)$ and $(3,5)$ cannot extend $\langle 1,3\rangle$, because

$$\max_{u \in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,3\rangle) + c^{\rho(u)}_{3i}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{i7}((G_5)^-_{\{2,3\}})\} = 4 \geq UB, \;\; i = 2,5.$$

But, for arc $(3,4)$, one has

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,3\rangle) + c^{\rho(u)}_{34}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{47}((G_5)^-_{\{2,3\}})\} = 1 < UB,$$

and, therefore, $(3,4)$ is added to $\langle 1,3\rangle$. The obtained path $\langle 1,3,4\rangle$ is included in X, which is updated to

$$X = \{\langle 1,2,5\rangle, \langle 1,3,4\rangle\},$$

and the label

$$z_{(G_5)^-_{\{2,3\}}}(\langle 1,3,4\rangle) = (-2,-1)$$

is set. The path $\langle 1,2,5\rangle$ is the next to be selected. From

$$Ad_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5\rangle \,|\, \tilde{p}) = \{(5,j) \in A : j \notin \{1,2\}\} = \{(5,3),(5,6),(5,7)\},$$

only arcs $(5,6)$ and $(5,7)$ are included in ST. In fact, for arc $(5,3)$,

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5\rangle) + c^{\rho(u)}_{53}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{37}((G_5)^-_{\{2,3\}})\} = 2 \geq UB.$$

However, for arc $(5,6)$,

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5\rangle) + c^{\rho(u)}_{56}((G_5)^-_{\{2,3\}}) + LB^{\rho(u)}_{67}((G_5)^-_{\{2,3\}})\} = 1 < UB,$$

which allows to include path $\langle 1,2,5,6\rangle$ in X,

$$X = \{\langle 1,3,4\rangle, \langle 1,2,5,6\rangle\},$$

and to determine the label

$$z_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5,6\rangle) = (-1,0).$$

For arc $(5,7)$, (4.4) is satisfied as

$$\begin{cases} RD^2_{G_5}(\langle 1,2,5,7\rangle) = 2 \geq RC_{G_5}(\tilde{p}), \text{ with } 2 \in S^* \\ \\ RCaux = \max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5\rangle) + c^{\rho(u)}_{57}((G_5)^-_{\{2,3\}})\} = 1 < UB \end{cases}$$

Consequently, $\langle 1,2,5,7\rangle$ is a new candidate for the robust shortest path of $(G_5)^-_{\{2,3\}}$ and, then, UB and sol are updated to

$$UB = 1 \quad \text{and} \quad sol = \langle 1,2,5,7\rangle.$$

With this update, all the paths in X,

$$X = \{\langle 1,3,4\rangle, \langle 1,2,5,6\rangle\},$$

must be tested in order to discard those that cannot produce optimal solutions in $(G_5)^-_{\{2,3\}}$, according to (4.2). In fact, none of them can be considered for extension, since

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,3,4\rangle) + LB^{\rho(u)}_{47}((G_5)^-_{\{2,3\}})\} = 1 \geq UB$$

and

$$\max_{u\in U_2}\{z^u_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5,6\rangle) + LB^{\rho(u)}_{67}((G_5)^-_{\{2,3\}})\} = 1 \geq UB.$$

Consequently, all the paths in X are eliminated and Algorithm 8 halts, returning $\langle 1,2,5,7\rangle$ as the robust shortest path in $(G_5)^-_{\{2,3\}}$, with $RC_{(G_5)^-_{\{2,3\}}}(\langle 1,2,5,7\rangle) = 1$.

Next, the reoptimization procedure for the extended version of G with respect to the set of scenarios is devised.

4.2.2 Addition of scenarios

Let $S^* = \{s_u : u \in U^*\}$, with $U^* = \{k+1,\ldots,k+k^*\}$, $k^* \geq 1$, be a finite set of scenarios added to G. The extended version of G, $G^+_{S^*}$, has set of scenarios $S \cup S^*$ and set of $(1,n)$-paths $P_{1n}(G^+_{S^*}) = P_{1n}(G)$. Like for the previous subsection, the results of Lemma 4.1 can be adapted, considering now the networks $G'_S = G$ and $G''_S = G^+_{S^*}$.

Corollary 4.4. *For any $p \in P_{1n}(G^+_{S^*}) = P_{1n}(G)$,*

1. *$RD^{s_u}_G(p) = RD^{s_u}_{G^+_{S^*}}(p)$, for any $u \in U_k$.*
2. *$RC_{G^+_{S^*}}(p) = \max\left\{RC_G(p)\,,\, \max_{u\in U^*} RD^{s_u}_{G^+_{S^*}}(p)\right\}$.*
3. *$RC_G(p) \leq RC_{G^+_{S^*}}(p)$.*

It should be noted that none of the robustness costs of the paths in $P_{1n}(G)$ decreases in $G^+_{S^*}$. The preservation of $\tilde{p}$ as a robust shortest path in $G^+_{S^*}$ depends on the scenarios added to S and on how they determine the relation of the associate robust deviations of $\tilde{p}$ with $RC_G(\tilde{p})$. In the following, conditions are established to assure such maintenance. Moreover, other possible optimal solutions can be restricted to a particular subset of $P_{1n}(G^+_{S^*})$.

Proposition 4.5. *Let $\widehat{U}^* \subseteq U^*$ be given by*

$$\widehat{U}^* = \{u \in U^* : RD^{s_u}_{G^+_{S^*}}(\tilde{p}) > RC_G(\tilde{p})\}.$$

1. *If $\widehat{U}^* = \emptyset$, then $\tilde{p}$ is a robust shortest path in $G^+_{S^*}$, with $RC_{G^+_{S^*}}(\tilde{p}) = RC_G(\tilde{p})$.*

2. *If* $\widehat{U}^* \neq \emptyset$, *then* $RC_{G^+_{S^*}}(\tilde{p}) = \max_{u \in \widehat{U}^*} RD^{s_u}_{G^+_{S^*}}(\tilde{p}) > RC_G(\tilde{p})$. *Let* $\widehat{P}_{1n}(G^+_{S^*}) \subseteq P_{1n}(G^+_{S^*})$ *be given by*

$$\widehat{P}_{1n}(G^+_{S^*}) = \{p \in P_{1n}(G^+_{S^*}) : RC_G(p) < RC_{G^+_{S^*}}(\tilde{p})\}.$$

   (a) *If* $p \in P_{1n}(G^+_{S^*})$ *satisfies* $RC_{G^+_{S^*}}(p) < RC_{G^+_{S^*}}(\tilde{p})$, *then* $p \in \widehat{P}_{1n}(G^+_{S^*})$.

   (b) $\tilde{p}$ *is a robust shortest path in* $G^+_{S^*}$ *if and only if* $RC_{G^+_{S^*}}(p) \geq RC_{G^+_{S^*}}(\tilde{p})$, *for any* $p \in \widehat{P}_{1n}(G^+_{S^*})$.

Proof.

1. If $\widehat{U}^* = \emptyset$, then, $RD^{s_u}_{G^+_{S^*}}(\tilde{p}) \leq RC_G(\tilde{p})$, for any $u \in U^*$, i.e.

$$\max_{u \in U^*} RD^{s_u}_{G^+_{S^*}}(\tilde{p}) \leq RC_G(\tilde{p}).$$

   Hence, from point 2. of Corollary 4.4 applied to $\tilde{p}$, one must have

$$RC_{G^+_{S^*}}(\tilde{p}) = RC_G(\tilde{p}).$$

   Because, by assumption, $\tilde{p}$ is a robust shortest path in G, $RC_G(p) \geq RC_G(\tilde{p})$, for any $p \in P_{1n}(G) = P_{1n}(G^+_{S^*})$, and, therefore,

$$RC_G(p) \geq RC_{G^+_{S^*}}(\tilde{p}), \text{ for any } p \in P_{1n}(G^+_{S^*}).$$

   Then, by point 3. of Corollary 4.4,

$$RC_{G^+_{S^*}}(p) \geq RC_{G^+_{S^*}}(\tilde{p}), \text{ for any } p \in P_{1n}(G^+_{S^*}),$$

   which means $\tilde{p}$ is a robust shortest path in $G^+_{S^*}$ and, moreover, it satisfies $RC_{G^+_{S^*}}(\tilde{p}) = RC_G(\tilde{p})$, as seen above.

2. If $\widehat{U}^* \neq \emptyset$, then, by definition of set $\widehat{U}^*$,

$$\max_{u \in U^*} RD^{s_u}_{G^+_{S^*}}(\tilde{p}) = \max_{u \in \widehat{U}^*} RD^{s_u}_{G^+_{S^*}}(\tilde{p}) > RC_G(\tilde{p}).$$

   Hence, by point 2. of Corollary 4.4 applied to $\tilde{p}$,

$$RC_{G^+_{S^*}}(\tilde{p}) = \max_{u \in \widehat{U}^*} RD^{s_u}_{G^+_{S^*}}(\tilde{p}) > RC_G(\tilde{p}).$$

   (a) Let $p \in P_{1n}(G^+_{S^*})$ satisfy $RC_{G^+_{S^*}}(p) < RC_{G^+_{S^*}}(\tilde{p})$. From point 3. of Corollary 4.4,

$$RC_G(p) < RC_{G^+_{S^*}}(\tilde{p}),$$

   and, therefore, $p \in \widehat{P}_{1n}(G^+_{S^*})$.

(b) The result is derived with the same reasoning applied for point 2. of Proposition 4.3, but now considering network $G^+_{S^*}$ and set $\widehat{P}_{1n}(G^+_{S^*})$.

□

According with Proposition 4.5, $\widehat{U}^*$ has to be known to evaluate the necessity of reoptimizing the problem at $G^+_{S^*}$. In case $\widehat{U}^* = \emptyset$, the method is skipped, as $\tilde{p}$ is still a robust shortest path in $G^+_{S^*}$. Otherwise, by setting the initial *sol* to $\tilde{p}$ and the initial UB to $RC_{G^+_{S^*}}(\tilde{p})$, a search is performed among the simple paths of $\widehat{P}_{1n}(G^+_{S^*})$. However, this set can be modified by considering the updates of UB as the upper-bounds of the robustness costs in G of the paths in $P_{1n}(G^+_{S^*})$, instead of considering $RC_{G^+_{S^*}}(\tilde{p})$. Hence, the new search set is defined by

$$\widehat{P}^{(UB)}_{1n}(G^+_{S^*}) = \{p \in P_{1n}(G^+_{S^*}) : RC_G(p) < UB\},$$

and it can be reduced whenever UB is updated. In fact, there is no need to analyze further any path $p \in P_{1n}(G^+_{S^*})$, such that $UB' \leq RC_G(p) < UB$, because, by point 3. of Corollary 4.4, under that condition, $RC_{G^+_{S^*}}(p) \geq UB'$. Hence, when updating UB, the search set is updated as well by $\widehat{P}^{(UB)}_{1n}(G^+_{S^*})$.

In the following, the extension rules for constructing ST for $G^+_{S^*}$ are described, according with the principles applied in the previous subsection to guide the search. For assigning labels, since k^* is the number of scenarios added to S, the label in $G^+_{S^*}$ for each $p_{1i} \in ST \cap P_{1i}(G^+_{S^*})$, $i \in V$, is a $(k+k^*)$-vector given by

$$z_{G^+_{S^*}}(p_{1i}) = (z^1_{G^+_{S^*}}(p_{1i}), \ldots, z^{k+k^*}_{G^+_{S^*}}(p_{1i})).$$

The first label is set for node 1 by

$$z_{G^+_{S^*}}(\langle 1 \rangle) = (-LB^{s_1}_{1n}(G), \ldots, -LB^{s_k}_{1n}(G), -LB^{s_{k+1}}_{1n}(G^+_{S^*}), \ldots, -LB^{s_{k+k^*}}_{1n}(G^+_{S^*})),$$

as $LB^{s_u}_{1n}(G^+_{S^*}) = LB^{s_u}_{1n}(G)$, for any $u \in U_k$.

Let $p_{1i} \in ST \cap P_{1i}(G^-_{S^*})$, $i \in V$, for which $(i,j) \in A$ is added, then, the label in $G^+_{S^*}$ for the $(1,j)$-path $p_{1j} = p_{1i} \diamond \langle i,j \rangle$, is

$$z_{G^+_{S^*}}(p_{1j}) = (z^1_{G^+_{S^*}}(p_{1i}) + c^{s_1}_{ij}(G^+_{S^*}), \ldots, z^{k+k^*}_{G^+_{S^*}}(p_{1i}) + c^{s_{k+k^*}}_{ij}(G^+_{S^*})).$$

When $j = n$,

$$z_{G^+_{S^*}}(p_{1n}) = (RD^{s_1}_{G^+_{S^*}}(p_{1n}), \ldots, RD^{s_{k+k^*}}_{G^+_{S^*}}(p_{1n})),$$

and, therefore,

$$RC_{G^+_{S^*}}(p_{1n}) = \max_{u \in U_{k+k^*}} z^u_{G^+_{S^*}}(p_{1n}).$$

Under these conditions, the first extension rule considers that $p_{1i} \in P_{1i}(G^+_{S^*})$, $i \in V\backslash\{n\}$, is part of ST if

$$\max_{u \in U_{k+k^*}} \{z^u_{G^+_{S^*}}(p_{1i}) + LB^{s_u}_{in}(G^+_{S^*})\} < UB. \tag{4.5}$$

The second rule considers that condition

$$\max_{u \in U_{k+k^*}} \{z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{S^*}) + LB^{s_u}_{jn}(G^+_{S^*})\} < UB. \tag{4.6}$$

must hold for any arc (i,j) that extends $p_{1i} \in P_{1i}(G^+_{S^*}) \cap ST$, $i \in V\backslash\{n\}$. As explained before, the part of $\tilde{p}$ included in ST is set in the initial list X, given by

$$X = \{\langle 1\rangle\} \cup \big\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in A(\tilde{p}),\ j \neq n, \text{ satisfy (4.6)}\}.$$

By knowing these paths, the subsequent extension technique applies to each $p_{1i} \in ST \cap P_{1i}(G^+_{S^*})$, $i \in V\backslash\{n\}$, by choosing the arcs in set $Ad_{G^+_{S^*}}(p_{1i}\,|\,\tilde{p})$, which is determined as in Algorithm 8. An arc $(i,j) \in Ad_{G^+_{S^*}}(p_{1i}\,|\,\tilde{p})$, with $j \neq n$, is added to p_{1i}, when (4.6) is verified. When $j = n$, the third extension rule applies to the obtained $(1,n)$-path, p_{1n}, which must belong to $\widehat{P}^{(UB)}_{1n}(G^+_{S^*})$ and must satisfy $RC_{G^+_{S^*}}(p_{1n}) < UB$. The labels in $G^+_{S^*}$ can be used to calculate $RC_G(p_{1n})$ and $RC_{G^+_{S^*}}(p_{1n})$, denoted by $aux1$ and $RCaux$, respectively. In fact,

$$aux1 = \max_{u \in U_k} RD^{s_u}_G(p_{1n}) = \max_{u \in U_k} RD^{s_u}_{G^+_{S^*}}(p_{1n}),$$

taking into account point 1. of Corollary 4.4, and

$$RCaux = \max\big\{aux1, \max_{u \in U^*} RD^{s_u}_{G^+_{S^*}}(p_{1n})\big\},$$

according to point 2. of Corollary 4.4, with

$$RD^{s_u}_{G^+_{S^*}}(p_{1n}) = z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{in}(G^+_{S^*}),\ u \in U_{k+k^*}.$$

Hence, if

$$\begin{cases} aux1 = \max_{u \in U_k}\{z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{in}(G^+_{S^*})\} < UB \\ \\ RCaux = \max\big\{aux1, \max_{u \in U^*}\{z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{in}(G^+_{S^*})\}\big\} < UB \end{cases} \tag{4.7}$$

are satisfied, the new candidate for the optimal solution, sol, is p_{1n} and UB is updated to $RCaux$. With this new value, the first extension rule must be tested for the paths in X. Only the paths satisfying (4.5) are considered for further extension, the remaining are discarded.

The pseudo-code of the reoptimization method is given in Algorithm 9.

Algorithm 9: Finding the robust shortest path in $G^+_{S^*}$, given $\tilde{p}$, $RC_G(\tilde{p})$ and $LB^{s_u}_{1n}(G)$, $u \in U_k$

for $u \in U^*$ **do** Compute $\mathcal{T}^{s_u}_n(G^+_{S^*})$ and $LB^{s_u}_{in}(G^+_{S^*})$, $i \in V$;
$sol \leftarrow \tilde{p}$; $\widehat{U}^* \leftarrow \{u \in U^* : RD^{s_u}_{G^+_{S^*}}(\tilde{p}) > RC_G(\tilde{p})\}$;
if $\widehat{U}^* \neq \emptyset$ **then**
  $RC_{G^+_{S^*}}(\tilde{p}) \leftarrow \max_{u \in \widehat{U}^*} RD^{s_u}_{G^+_{S^*}}(\tilde{p})$; $UB \leftarrow RC_{G^+_{S^*}}(\tilde{p})$; $X \leftarrow \{\langle 1 \rangle\}$;
  for $u \in U_k$ **do** Compute $\mathcal{T}^{s_u}_n(G^+_{S^*})$ and $LB^{s_u}_{in}(G^+_{S^*})$, $i \in V$;
  for $u \in U_{k+k^*}$ **do** $z^u_{G^+_{S^*}}(\langle 1 \rangle) \leftarrow -LB^{s_u}_{1n}(G^+_{S^*})$;
  for $(i,j) \in A(\tilde{p})$ ***and*** $j \neq n$ **do**
    if $\max_{u \in U_{k+k^*}} \{z^u_{G^+_{S^*}}(\tilde{p}_{1i}) + c^{s_u}_{ij}(G^+_{S^*}) + LB^{s_u}_{jn}(G^+_{S^*})\} < UB$ **then**
      $X \leftarrow X \cup \{\tilde{p}_{1j}\}$;
      for $u \in U_{k+k^*}$ **do** $z^u_{G^+_{S^*}}(\tilde{p}_{1j}) \leftarrow z^u_{G^+_{S^*}}(\tilde{p}_{1i}) + c^{s_u}_{ij}(G^+_{S^*})$;
    else break;
  while $X \neq \emptyset$ **do**
    $p_{1i} \leftarrow$ first path in X; $X \leftarrow X - \{p_{1i}\}$; Compute $Ad_{G^+_{S^*}}(p_{1i} \,|\, \tilde{p})$;
    for $(i,j) \in Ad_{G^+_{S^*}}(p_{1i} \,|\, \tilde{p})$ **do**
      if $j = n$ **then**
        $aux1 \leftarrow \max_{u \in U_k} \{z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{S^*})\}$;
        if $aux1 < UB$ **then**
          $RCaux \leftarrow \max\{aux1, \max_{u \in U^*} \{z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{S^*})\}\}$;
          if $RCaux < UB$ **then**
            $UB \leftarrow RCaux$; $sol \leftarrow p_{1i} \diamond \langle i,j \rangle$;
            for $p_{1i'} \in X$ **do**
              for $u \in U_{k+k^*}$ **do**
                if $z^u_{G^+_{S^*}}(p_{1i'}) + LB^{s_u}_{i'n}(G^+_{S^*}) \geq UB$ **then**
                  $X \leftarrow X - \{p_{1i'}\}$; **break**;
      else
        if $\max_{u \in U_{k+k^*}} \{z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{S^*}) + LB^{s_u}_{jn}(G^+_{S^*})\} < UB$ **then**
          $p_{1j} \leftarrow p_{1i} \diamond \langle i,j \rangle$; $X \leftarrow X \cup \{p_{1j}\}$;
          for $u \in U_{k+k^*}$ **do** $z^u_{G^+_{S^*}}(p_{1j}) \leftarrow z^u_{G^+_{S^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{S^*})$;
return sol;

Computational time complexity order Let $\bar{k} = k + k^*$ denote the number of scenarios in $G^+_{S^*}$ and W^+_S denote the maximum number of paths generated in $ST \cap P_{1i}(G^+_{S^*})$, $i \in V$. Algorithm 9 has two phases. The first regards the preliminary computations to perform the search for a robust shortest path in $G^+_{S^*}$. The second is concerned with the latter task for the extension and labeling of the paths in ST.

In a worst case, the first phase demands the calculation of the trees $\mathcal{T}^{s_u}_n(G^+_{S^*})$ and of the costs $LB^{s_u}_{in}(G^+_{S^*})$, $i \in V$, $u \in U_{\bar{k}}$, in $O^a_1 = \mathcal{O}(\bar{k}m)$ time for acyclic networks and in $O^c_1 = \mathcal{O}(\bar{k}(m+n\log n))$ for general networks, as seen in Algorithm 8, but now considering $\bar{k}$ scenarios. To determine set $\widehat{U}^*$ and, later, in a worst case, the initial UB, given by $RC_{G^+_{S^*}}(\tilde{p})$, $\mathcal{O}(k^*n)$ operations are required. Nevertheless, this complexity does not modify the previous.

The second phase concerns the development of ST. The generation of each label and the application of the extension rule (4.6) demand $\mathcal{O}(\bar{k})$ time. These operations are needed to set the part of $\tilde{p}$ included in ST and, later, when extending the paths by means of the **while** loop in line 12, which requires $(n-1)W^+_S$ iterations at most. Each of them, implies at most $n-1$ iterations of the **for** loop in line 14, when selecting the arcs to be evaluated. The remaining operations in each of those iterations are the extension rules (4.7) and (4.5). Both can be done for each path in $\mathcal{O}(\bar{k})$ time. Whenever UB is updated, (4.5) must be repeated in the **for** loop of line 21, for every path in list X, which has at most $(n-1)W^+_S$ elements, and therefore an $\mathcal{O}(\bar{k}nW^+_S)$ complexity is demanded. Consequently, the second phase is performed in $O_2 = \mathcal{O}(\bar{k}n^3(W^+_S)^2)$.

In conclusion, Algorithm 9 has a total time complexity of $\mathcal{O}(\bar{k}n^3(W^+_S)^2)$ for any type of network, given that $\log n \ll n$ and $m < n^2$.

Examples Next, two examples of how to reoptimize the robust shortest path problem in network G_5 – Figure 4.1, are provided, by extending G_5 with respect to its set of scenarios. One illustrates $\widehat{U}^* = \emptyset$ and the other the opposite case. In the first example, only one scenario is added to G_5 and, in the second, another scenario is added to the first example. For each, a simple robust shortest path is determined by Algorithm 9.

Recall that for G_5, $\tilde{p} = \langle 1, 4, 6, 7\rangle$, with $RC_{G_5}(\tilde{p}) = 2$.

Case 1 The network obtained from G_5 after the addition of scenario 5, $(G_5)^+_5$, is depicted in Figure 4.5. For the new scenario, $p^{1,5}((G_5)^+_5) = \langle 1, 3, 5, 7\rangle$, with $LB^5_{17}((G_5)^+_5) = 2$. Hence,

$$RD^5_{(G_5)^+_5}(\tilde{p}) = c^5_{(G_5)^+_5}(\tilde{p}) - LB^5_{17}((G_5)^+_5) = 2 \leq RC_{G_5}(\tilde{p}), \tag{4.8}$$

which means,

$$\widehat{U}^* = \emptyset.$$

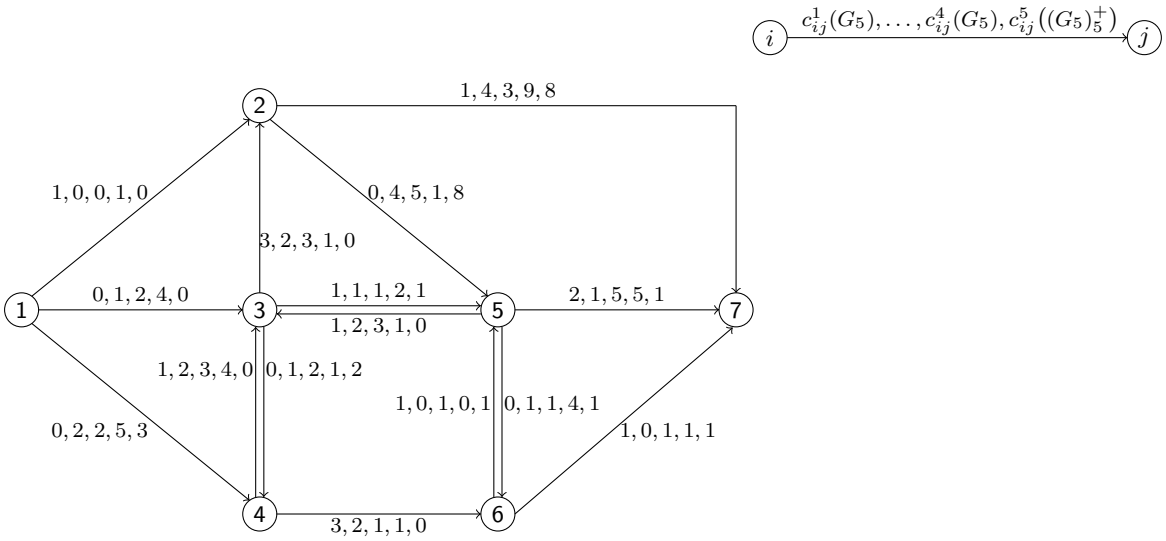


Figure 4.5: Network $(G_5)_5^+$

Consequently, Algorithm 9 returns $\tilde{p}$ as the robust shortest path of $(G_5)_5^+$, for which $RC_{(G_5)_5^+}(\tilde{p}) = RC_{G_5}(\tilde{p}) = 2$, by 1. of Proposition 4.5.

Case 2 Assume now that two scenarios are added to G_5 – Figure 4.6, with scenario 5 defined like in Case 1. In this example, $\mathcal{T}_7^u((G_5)_{\{5,6\}}^+) = \mathcal{T}_7^u(G_5)$, $u \in U_4$. For scenarios 1 and 4, these trees were already represented in Figure 4.3. For the remaining scenarios, the trees are represented in Figure 4.7. Since scenario 5 is defined like in Case 1, condition (4.8) is satisfied and also

$$RD^6_{(G_5)^+_{\{5,6\}}}(\tilde{p}) = c^6_{(G_5)^+_{\{5,6\}}}(\tilde{p}) - LB^6_{17}((G_5)^+_{\{5,6\}}) = 6 > RC_G(\tilde{p}),$$

which allows to derive that

$$\widehat{U}^* = \{6\}.$$

As $\widehat{U}^* \neq \emptyset$, UB and sol are initialized by

$$UB = RC_{(G_5)^+_{\{5,6\}}}(\tilde{p}) = \max\left\{RC_{G_5}(\tilde{p}), \max_{u\in\{5,6\}} RD^u_{(G_5)^+_{\{5,6\}}}(\tilde{p})\right\} = 6 \quad \text{and} \quad sol = \tilde{p} = \langle 1,4,6,7\rangle.$$

Consequently, ST has to be developed from node 1, with label

$$z_{(G_5)^+_{\{5,6\}}}(\langle 1\rangle) = (-2,-3,-3,-6,-2,-7),$$

and with the sub-paths of $\tilde{p}$ in

$$X = \left\{\langle 1\rangle\right\} \cup \left\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in \{(1,4),(4,6)\} \text{ satisfy (4.6)}\right\}.$$

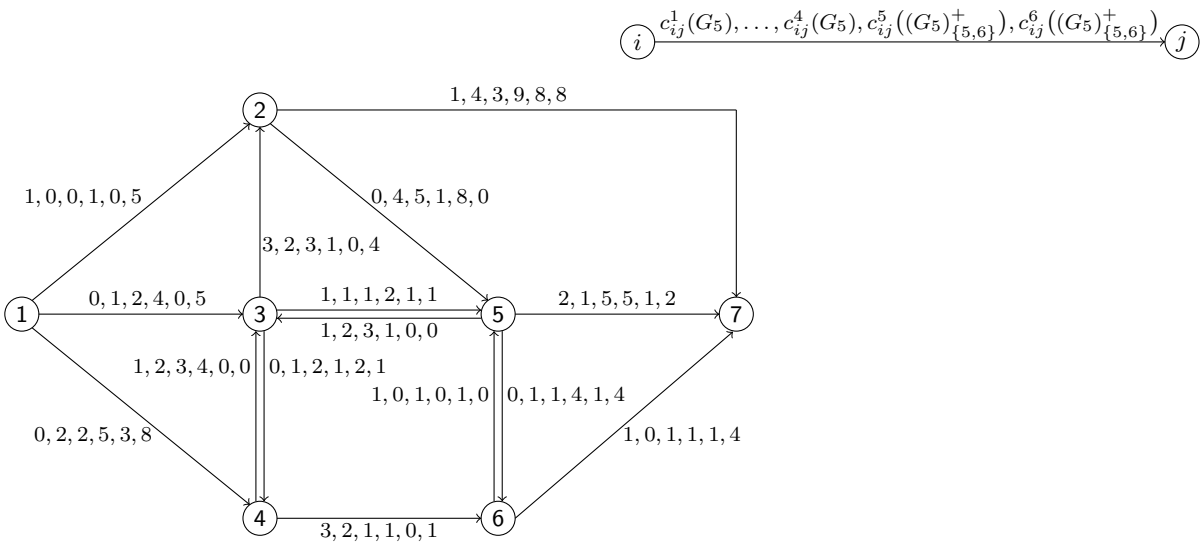


Figure 4.6: Network $(G_5)^+_{\{5,6\}}$

Path $\langle 1, 4\rangle$ belongs to ST, because

$$\max_{u \in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1\rangle) + c^u_{14}((G_5)^+_{\{5,6\}}) + LB^u_{47}((G_5)^+_{\{5,6\}})\} = 4 < UB,$$

which allows to set

$$z_{(G_5)^+_{\{5,6\}}}(\langle 1, 4\rangle) = (-2, -1, -1, -1, 1, 1).$$

Then, arc $(4, 6)$ can be added to $\langle 1, 4\rangle$, since

$$\max_{u \in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1, 4\rangle) + c^u_{46}((G_5)^+_{\{5,6\}}) + LB^u_{67}((G_5)^+_{\{5,6\}})\} = 4 < UB,$$

and the next label is

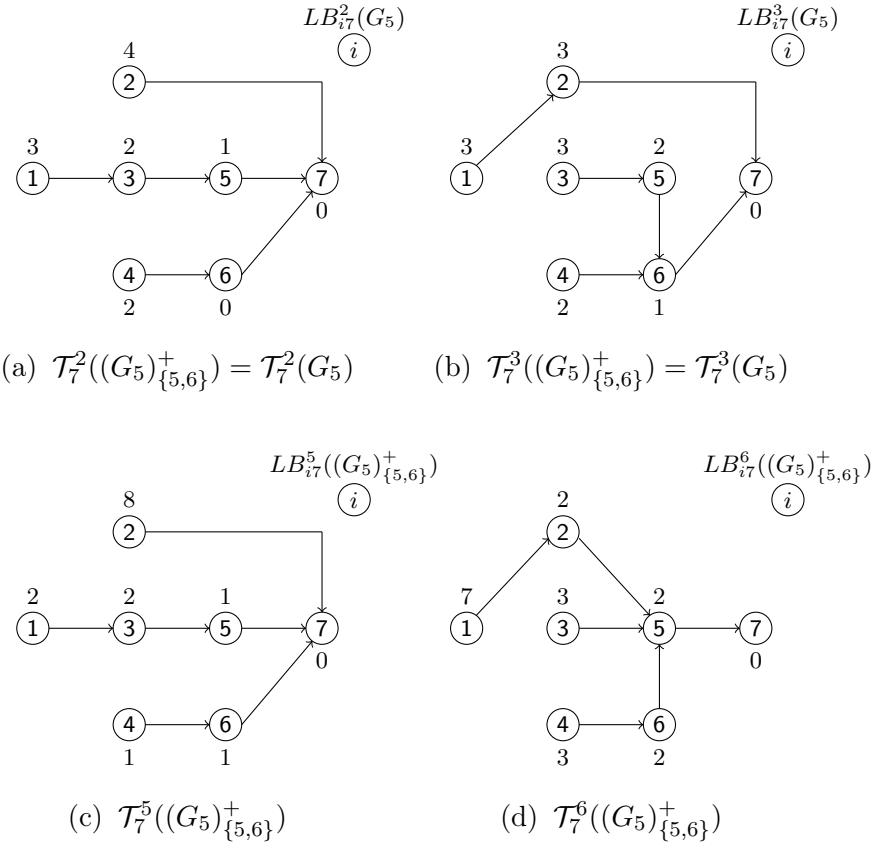
$$z_{(G_5)^+_{\{5,6\}}}(\langle 1, 4, 6\rangle) = (1, 1, 0, 0, 1, 2).$$

Therefore, ST starts with the paths of

$$X = \{\langle 1\rangle, \langle 1, 4\rangle, \langle 1, 4, 6\rangle\}.$$

Figure 4.8 shows the ST resultant from applying Algorithm 9 to $(G_5)^+_{\{5,6\}}$. The arcs available for extending $\langle 1\rangle$ belong to

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1\rangle \,|\, \tilde{p}) = \{(1, j) \in A \backslash A(\tilde{p})\} = \{(1, 2), (1, 3)\}.$$



(a) $\mathcal{T}_7^2((G_5)^+_{\{5,6\}}) = \mathcal{T}_7^2(G_5)$ (b) $\mathcal{T}_7^3((G_5)^+_{\{5,6\}}) = \mathcal{T}_7^3(G_5)$

(c) $\mathcal{T}_7^5((G_5)^+_{\{5,6\}})$ (d) $\mathcal{T}_7^6((G_5)^+_{\{5,6\}})$

Figure 4.7: Shortest path trees rooted at node 7 for scenarios 2 and 3 in G_5 and $(G_5)^+_{\{5,6\}}$, and for scenarios 5 and 6 in $(G_5)^+_{\{5,6\}}$

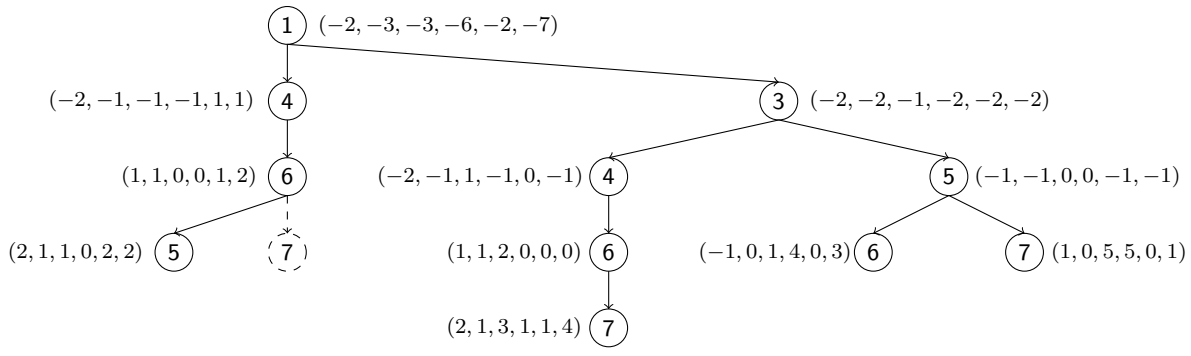


Figure 4.8: ST produced by Algorithm 9 for $(G_5)^+_{\{5,6\}}$

For arc $(1,2)$,

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1\rangle) + c^u_{12}((G_5)^+_{\{5,6\}}) + LB^u_{27}((G_5)^+_{\{5,6\}})\} = 6 \geq UB$$

and, for arc $(1,3)$,

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1\rangle) + c^u_{13}((G_5)^+_{\{5,6\}}) + LB^u_{37}((G_5)^+_{\{5,6\}})\} = 2 < UB.$$

Consequently, X is updated to

$$X = \{\langle 1,4\rangle, \langle 1,4,6\rangle, \langle 1,3\rangle\},$$

and the label

$$z_{(G_5)^+_{\{5,6\}}}(\langle 1,3\rangle) = (-2,-2,-1,-2,-2,-2)$$

is obtained. For the next iteration, path $\langle 1,4\rangle$ is selected for extension. In this case, the arc in

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1,4\rangle \,|\, \tilde{p}) = \{(4,j) \in A\backslash A(\tilde{p}) : j \neq 1\} = \{(4,3)\}$$

cannot be added to $\langle 1,4\rangle$, because

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,4\rangle) + c^u_{43}((G_5)^+_{\{5,6\}}) + LB^u_{37}((G_5)^+_{\{5,6\}})\} = 6 \geq UB.$$

Afterwards, path $\langle 1,4,6\rangle$ is selected for scanning with

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1,4,6\rangle) = \{(6,j) \in A\backslash A(\tilde{p}) : j \notin \{1,4\}\} = \{(6,5)\}.$$

The arc of this set extends $\langle 1,4,6\rangle$, since

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,4,6\rangle) + c^u_{65}((G_5)^+_{\{5,6\}}) + LB^u_{57}((G_5)^+_{\{5,6\}})\} = 4 < UB.$$

Then, X is updated to

$$X = \{\langle 1,3\rangle, \langle 1,4,6,5\rangle\},$$

and the label

$$z_{(G_5)^+_{\{5,6\}}}(\langle 1,4,6,5\rangle) = (2,1,1,0,2,2).$$

is obtained. Path $\langle 1,3\rangle$ is the next in X to be evaluated. In this case,

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1,3\rangle \,|\, \tilde{p}) = \{(3,j) \in A : j \neq 1\} = \{(3,2),(3,4),(3,5)\}.$$

Arc $(3,2)$ cannot extend $\langle 1,3\rangle$, since

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3\rangle) + c^u_{32}((G_5)^+_{\{5,6\}}) + LB^u_{27}((G_5)^+_{\{5,6\}})\} = 6 \geq UB.$$

Nevertheless, arcs $(3,4)$ and $(3,5)$ may extend $\langle 1,3\rangle$, because

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3\rangle) + c^u_{34}((G_5)^+_{\{5,6\}}) + LB^u_{47}((G_5)^+_{\{5,6\}})\} = 3 < UB$$

and

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3\rangle) + c^u_{35}((G_5)^+_{\{5,6\}}) + LB^u_{57}((G_5)^+_{\{5,6\}})\} = 4 < UB.$$

Now, X is updated to

$$X = \{\langle 1,4,6,5\rangle, \langle 1,3,4\rangle, \langle 1,3,5\rangle\}$$

and the labels

$$z_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4\rangle) = (-2,-1,1,-1,0,-1) \quad \text{and} \quad z_{(G_5)^+_{\{5,6\}}}(\langle 1,3,5\rangle) = (-1,-1,0,0,-1,-1)$$

are determined. Next, picking $\langle 1,4,6,5\rangle$ in X to be extended, one has

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1,4,6,5\rangle \,|\, \tilde{p}) = \{(5,j) \in A : j \notin \{1,4,6\}\} = \{(5,7)\}.$$

For arc $(5,7)$, the first condition of (4.7) is not satisfied, because

$$aux1 = \max_{u\in U_4}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,4,6,5\rangle) + c^u_{57}((G_5)^+_{\{5,6\}})\} = 6 \geq UB.$$

Therefore, arc $(5,7)$ is not included in ST. Path $\langle 1,3,4\rangle$ is the next element of X for possible extension, with

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4\rangle \,|\, \tilde{p}) = \{(4,j) \in A : j \notin \{1,3\}\} = \{(4,6)\}.$$

The arc of this set is added to $\langle 1,3,4\rangle$, since

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4\rangle) + c^u_{46}((G_5)^+_{\{5,6\}}) + LB^u_{67}((G_5)^+_{\{5,6\}})\} = 3 < UB.$$

Then, X is updated to

$$X = \{\langle 1,3,5\rangle, \langle 1,3,4,6\rangle\}$$

and the label

$$z_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4,6\rangle) = (1,1,2,0,0,0)$$

is established. Afterwards, path $\langle 1,3,5\rangle$ is scanned, with

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1,3,5\rangle \,|\, \tilde{p}) = \{(5,j) \in A : j \notin \{1,3\}\} = \{(5,6),(5,7)\}.$$

All the arcs in this set can be added to $\langle 1,3,5\rangle$. In fact, for arc $(5,6)$,

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,5\rangle) + c^u_{56}((G_5)^+_{\{5,6\}}) + LB^u_{67}((G_5)^+_{\{5,6\}})\} = 5 < UB,$$

with the label for $\langle 1,3,5,6\rangle$ given by

$$z_{(G_5)^+_{\{5,6\}}}(\langle 1,3,5,6\rangle) = (-1,0,1,4,0,3)$$

and list X updated to

$$X = \{\langle 1,3,4,6\rangle, \langle 1,3,5,6\rangle\}.$$

For arc $(5,7)$, the two conditions of (4.7) are satisfied since

$$\begin{cases} aux1 = \max_{u\in U_4}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,5\rangle) + c^u_{57}((G_5)^+_{\{5,6\}})\} = 5 < UB \\ \\ RCaux = \max\left\{aux1, \max_{u\in\{5,6\}}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,5\rangle) + c^u_{57}((G_5)^+_{\{5,6\}})\}\right\} = 5 < UB \end{cases}$$

Then, $\langle 1,3,5,7\rangle$ is a new candidate for an optimal solution in $(G_5)^+_{\{5,6\}}$, and UB and sol are updated to

$$UB = 5 \quad \text{and} \quad sol = \langle 1,3,5,7\rangle.$$

Afterwards, all the paths in X must be tested. Path $\langle 1,3,4,6\rangle$ cannot be discarded, as

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4,6\rangle) + LB^u_{67}((G_5)^+_{\{5,6\}})\} = 3 < UB.$$

Nevertheless, for $\langle 1,3,5,6\rangle$,

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,5,6\rangle) + LB^u_{67}((G_5)^+_{\{5,6\}})\} = 5 \geq UB,$$

and, therefore, $\langle 1,3,5,6\rangle$ is deleted from X, remaining only $\langle 1,3,4,6\rangle$ to analyze. One has

$$Ad_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4,6\rangle \,|\, \tilde{p}) = \{(6,j) \in A : j \notin \{1,3,4\}\} = \{(6,5),(6,7)\}.$$

It can be concluded that arc $(6,5)$ cannot extend $\langle 1,3,4,6\rangle$, since

$$\max_{u\in U_6}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4,6\rangle) + c^u_{65}((G_5)^+_{\{5,6\}}) + LB^u_{57}((G_5)^+_{\{5,6\}})\} = 5 \geq UB,$$

however, arc $(6,7)$ can extend $\langle 1,3,4,6\rangle$, since (4.7) holds, with

$$\begin{cases} aux1 = \max_{u\in U_4}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4,6\rangle) + c^u_{67}((G_5)^+_{\{5,6\}})\} = 3 < UB \\ \\ RCaux = \max\left\{aux1, \max_{u\in\{5,6\}}\{z^u_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4,6\rangle) + c^u_{67}((G_5)^+_{\{5,6\}})\}\right\} = 4 < UB. \end{cases}$$

Consequently, UB and sol are updated to

$$UB = 4 \quad \text{and} \quad sol = \langle 1,3,4,6,7\rangle.$$

Because there are no other paths in X to extend, Algorithm 9 returns $\langle 1,3,4,6,7\rangle$ as the robust shortest path in $(G_5)^+_{\{5,6\}}$, with $RC_{(G_5)^+_{\{5,6\}}}(\langle 1,3,4,6,7\rangle) = 4$.

In the following, the reoptimization methods for the reduced and extended versions of G in terms of its set of arcs are going to be treated.

4.3 Variation of the number of arcs

This section is dedicated to develop the methods for reoptimizing the robust shortest path when arcs are deleted from, or inserted in G. Analogously to the previous section, the first goal is to establish the conditions required for $\tilde{p}$ preserving its optimality in the obtained reduced and extended versions of G. A general result is first introduced for networks that differ from each other in terms of dimension with respect to the set of arcs and nodes. In specific, assume that two networks have the same set of scenarios, and that the sets of nodes and arcs of one of them is a subset of the other. Under these conditions, the set of paths common to both networks is the set of paths in the smallest of them. Moreover, the shortest $(1,n)$-paths of the networks for the same scenarios may differ and have different costs. This fact may affect the relations between the robust deviations for the same scenarios and between the robustness costs of the $(1,n)$-paths common to both networks as the next result shows.

Lemma 4.6. *Let $G'_A = G'_A(V', A', S)$ and $G''_A = G''_A(V'', A'', S)$ be connected networks, such that $A' \subseteq A''$, $V' \subseteq V''$ and $S = \{s_u : u \in U_k\}$ is a finite set. Let*

$$\widehat{U}_{A'}^{A''} = \{u \in U_k : LB_{1n}^{s_u}(G''_A) < LB_{1n}^{s_u}(G'_A)\}$$

be the set of scenarios indices for which the shortest $(1,n)$-path in G''_A is cheaper than the homologous path in G'_A. Then, $P_{1n}(G'_A) \subseteq P_{1n}(G''_A)$, and, for any $p \in P_{1n}(G'_A)$,

1. *$RD_{G'_A}^{s_u}(p) = RD_{G''_A}^{s_u}(p) + LB_{1n}^{s_u}(G''_A) - LB_{1n}^{s_u}(G'_A)$, for any $u \in U_k$. In particular, $RD_{G'_A}^{s_u}(p) = RD_{G''_A}^{s_u}(p)$, for any $u \in U_k \backslash \widehat{U}_{A'}^{A''}$.*
2. *$RC_{G'_A}(p) \leq RC_{G''_A}(p)$. In particular, if $\widehat{U}_{A'}^{A''} = \emptyset$, then $RC_{G'_A}(p) = RC_{G''_A}(p)$.*

Proof. Because $A' \subseteq A''$, $P_{1n}(G'_A) \subseteq P_{1n}(G''_A)$ immediately follows. Consequently,

$$c_{G'_A}^{s_u}(p) = c_{G''_A}^{s_u}(p)\,, \text{ for any } p \in P_{1n}(G'_A) \text{ and } u \in U_k. \tag{4.9}$$

Therefore, for any scenario s_u, $u \in U_k$, the cost of a shortest $(1,n)$-path in G''_A is never worse than the cost of the homologous path in G'_A, i.e.

$$LB_{1n}^{s_u}(G''_A) \leq LB_{1n}^{s_u}(G'_A)\,, \text{ for any } u \in U_k. \tag{4.10}$$

Let p be any path in $P_{1n}(G'_A)$.

1. Let $u \in U_k$. By definition of robust deviation for scenario s_u in G'_A,

$$RD_{G'_A}^{s_u}(p) = c_{G'_A}^{s_u}(p) - LB_{1n}^{s_u}(G'_A) = c_{G''_A}^{s_u}(p) - LB_{1n}^{s_u}(G'_A),$$

according with (4.9). Then, summing and subtracting $LB_{1n}^{s_u}(G''_A)$ results in

$$RD_{G'_A}^{s_u}(p) = RD_{G''_A}^{s_u}(p) + LB_{1n}^{s_u}(G''_A) - LB_{1n}^{s_u}(G'_A),$$

by definition of robust deviation for scenario s_u in G''_A. Now, from the definition of set $\widehat{U}_{A'}^{A''}$ and (4.10), for any $u \in U_k \backslash \widehat{U}_{A'}^{A''}$,

$$LB_{1n}^{s_u}(G'_A) = LB_{1n}^{s_u}(G''_A),$$

and, therefore,

$$RD_{G'_A}^{s_u}(p) = RD_{G''_A}^{s_u}(p)\,, \text{ for any } u \in U_k \backslash \widehat{U}_{A'}^{A''}.$$

2. From (4.9), (4.10) and the definition of robust deviation,

$$RD_{G'_A}^{s_u}(p) \leq RD_{G''_A}^{s_u}(p)\,, \text{ for any } u \in U_k.$$

Therefore, by the definition of robustness cost,

$$RC_{G'_A}(p) \leq RC_{G''_A}(p).$$

Now, if $\widehat{U}_{A'}^{A''} = \emptyset$, then, by point 1.,

$$RD_{G'_A}^{s_u}(p) = RD_{G''_A}^{s_u}(p)\,, \text{ for any } u \in U_k,$$

and, thus,

$$RC_{G'_A}(p) = RC_{G''_A}(p).$$

□

Lemma 4.6 gives auxiliary conditions to derive results for preserving, or not, the optimality of $\tilde{p}$ in the reduced and extended versions of G with respect to the set of arcs. In the following, the topics analyzed in the previous section are approached by the same order, adapted to the new modifications. To begin, the deletion of arcs from G is studied.

4.3.1 Elimination of arcs

Let A^* represent the set of arcs removed from A in G, such that $G^-_{A^*}$ is still a connected network. Under these conditions, some nodes of G may have to be removed as well, except 1 and n. Network $G^-_{A^*}$ has set of arcs $A \backslash A^*$ and set of $(1,n)$-paths $P_{1n}(G^-_{A^*}) \subseteq P_{1n}(G)$. Corollary 4.7 is a consequence of Lemma 4.6, by considering the networks $G'_A = G^-_{A^*}$ and $G''_A = G$.

Corollary 4.7. *For any $p \in P_{1n}(G^-_{A^*})$,*

1. $RD^{s_u}_{G^-_{A^*}}(p) = RD^{s_u}_G(p) + LB^{s_u}_{1n}(G) - LB^{s_u}_{1n}(G^-_{A^*})$, *for any $u \in U_k$. In particular,* $RD^{s_u}_{G^-_{A^*}}(p) = RD^{s_u}_G(p)$, *for any $u \in U_k \setminus \widehat{U}^A_{A\setminus A^*}$.*

2. $RC_{G^-_{A^*}}(p) \leq RC_G(p)$. *In particular, if $\widehat{U}^A_{A\setminus A^*} = \emptyset$, then $RC_{G^-_{A^*}}(p) = RC_G(p)$.*

Assuming that none of the arcs of $\tilde{p}$ is removed from A, the next result takes into account set $\widehat{U}^A_{A\setminus A^*}$ to derive conditions under which $\tilde{p}$ is a robust shortest path in $G^-_{A^*}$. Otherwise, a subset of $P_{1n}(G^-_{A^*})$ is provided, containing other possible optimal solutions.

Proposition 4.8. *Let $A(\tilde{p}) \subseteq A\setminus A^*$.*

1. *If $\widehat{U}^A_{A\setminus A^*} = \emptyset$, then $\tilde{p}$ is a robust shortest path in $G^-_{A^*}$, with $RC_{G^-_{A^*}}(\tilde{p}) = RC_G(\tilde{p})$.*

2. *If $\widehat{U}^A_{A\setminus A^*} \neq \emptyset$, let $\widehat{P}_{1n}(G^-_{A^*}) \subseteq P_{1n}(G^-_{A^*})$ be given by*

$$\widehat{P}_{1n}(G^-_{A^*}) = \{p \in P_{1n}(G^-_{A^*}) : RD^{s_u}_G(p) \geq RC_G(\tilde{p}),\ \textit{for some } u \in \widehat{U}^A_{A\setminus A^*}\}.$$

   (a) *If $p \in P_{1n}(G^-_{A^*})$ satisfies $RC_{G^-_{A^*}}(p) < RC_{G^-_{A^*}}(\tilde{p})$, then $p \in \widehat{P}_{1n}(G^-_{A^*})$.*

   (b) *$\tilde{p}$ is a robust shortest path in $G^-_{A^*}$ if and only if $RC_{G^-_{A^*}}(p) \geq RC_{G^-_{A^*}}(\tilde{p})$, for any $p \in \widehat{P}_{1n}(G^-_{A^*})$.*

Proof. Let $A(\tilde{p}) \subseteq A\setminus A^*$, which means that $\tilde{p} \in P_{1n}(G^-_{A^*})$. Since $\tilde{p}$ is a robust shortest path in G and $P_{1n}(G^-_{A^*}) \subseteq P_{1n}(G)$,

$$RC_G(p) \geq RC_G(\tilde{p}), \text{ for any } p \in P_{1n}(G^-_{A^*}). \tag{4.11}$$

1. If $\widehat{U}^A_{A\setminus A^*} = \emptyset$, by point 2. of Corollary 4.7, (4.11) can be rewritten as

$$RC_{G^-_{A^*}}(p) \geq RC_{G^-_{A^*}}(\tilde{p}), \text{ for any } p \in P_{1n}(G^-_{A^*}),$$

   with

$$RC_{G^-_{A^*}}(\tilde{p}) = RC_G(\tilde{p}).$$

   Consequently, $\tilde{p}$ is a robust shortest path in $G^-_{A^*}$ satisfying the condition above.

2. Assume that $\widehat{U}^A_{A\setminus A^*} \neq \emptyset$.

   (a) For any $p \in P_{1n}(G^-_{A^*}) \subseteq P_{1n}(G)$, the definition of robustness cost in G gives

$$RC_G(p) = \max\big\{ \max_{u\in\widehat{U}^A_{A\setminus A^*}} RD^{s_u}_G(p), \max_{u\in U_k\setminus\widehat{U}^A_{A\setminus A^*}} RD^{s_u}_G(p)\big\}. \tag{4.12}$$

with

$$\max_{u\in U_k \setminus \widehat{U}^A_{A\setminus A^*}} RD^{s_u}_G(p) = \max_{u\in U_k \setminus \widehat{U}^A_{A\setminus A^*}} RD^{s_u}_{G^-_{A^*}}(p) \leq \max_{u\in U_k} RD^{s_u}_{G^-_{A^*}}(p) = RC_{G^-_{A^*}}(p),$$

by point 1. of Corollary 4.7 for the first equality and the definition of robustness cost in $G^-_{A^*}$ for the last equality. Now, assume that $RC_{G^-_{A^*}}(p) < RC_{G^-_{A^*}}(\tilde{p})$. Then, from the condition above,

$$\max_{u\in U_k \setminus \widehat{U}^A_{A\setminus A^*}} RD^{s_u}_G(p) < RC_{G^-_{A^*}}(\tilde{p}).$$

Moreover, point 2. of Corollary 4.7 applied to $\tilde{p}$ and (4.11) results in

$$RC_{G^-_{A^*}}(\tilde{p}) \leq RC_G(p).$$

Therefore,

$$\max_{u\in U_k \setminus \widehat{U}^A_{A\setminus A^*}} RD^{s_u}_G(p) < RC_G(p),$$

which, from (4.12) and (4.11), allows to write

$$RC_G(p) = \max_{u\in \widehat{U}^A_{A\setminus A^*}} RD^{s_u}_G(p) \geq RC_G(\tilde{p}).$$

Therefore,

$$RD^{s_u}_G(p) \geq RC_G(\tilde{p}), \text{ for some } u \in \widehat{U}^A_{A\setminus A^*},$$

i.e., $p \in \widehat{P}_{1n}(G^-_{A^*})$.

(b) The proof is analogous to the presented for points 2. of Proposition 4.3 and 2.(b) of Proposition 4.5, by considering the network $G^-_{A^*}$ and the set $\widehat{P}_{1n}(G^-_{A^*})$.

□

According to Proposition 4.8, if $A(\tilde{p}) \cap A^* = \emptyset$, set $\widehat{U}^A_{A\setminus A^*}$ must be determined first. In case it is empty, $\tilde{p}$ is a robust shortest path of $G^-_{A^*}$, otherwise, the search for an optimal solution of $G^-_{A^*}$ must be performed over the simple paths of set $\widehat{P}_{1n}(G^-_{A^*})$ with a robustness cost in $G^-_{A^*}$ that can improve the best achieved. In this case, UB and *sol* can be initialized with $RC_{G^-_{A^*}}(\tilde{p})$ and $\tilde{p}$, respectively.

Instead, if $A(\tilde{p}) \cap A^* \neq \emptyset$, then $\tilde{p} \notin P(G^-_{A^*})$, and, therefore, there is a different robust shortest path in $G^-_{A^*}$. Under these conditions, like for the algorithms introduced in Chapter 2, the first candidate to be an optimal solution of $G^-_{A^*}$ is chosen among the shortest $(1,n)$-paths in $G^-_{A^*}$ for the scenarios of S with the least robustness cost in $G^-_{A^*}$. Therefore, UB and

sol are initialized with $\min\{RC_{G^-_{A^*}}(p^{1,s_u}(G^-_{A^*})) : u \in U_k\}$ and with path $p^{1,s_{u'}}(G^-_{A^*})$ such that $RC_{G^-_{A^*}}(p^{1,s_{u'}}(G^-_{A^*})) = UB$, for some $u' \in U_k$, respectively. In this case, because Proposition 4.8 does not apply, the search must take into consideration the simple paths in $P_{1n}(G^-_{A^*})$.

The construction of ST is justified when there is no guarantee that $\tilde{p}$ is a robust shortest path of $G^-_{A^*}$, that is, when $UB \neq 0$ and $\widehat{U}^A_{A\backslash A^*} \neq \emptyset$, if $A(\tilde{p}) \cap A^* = \emptyset$, or when $UB \neq 0$, otherwise.

In the following, the extension rules for the paths in ST are described. The labels are generated as in Algorithm 1, with k scenarios, starting with

$$z_{G^-_{A^*}}(\langle 1\rangle) = (-LB^{s_1}_{1n}(G^-_{A^*}), \ldots, -LB^{s_k}_{1n}(G^-_{A^*})).$$

Then, the first and the second extension rules apply analogously to Section 4.1. Namely, for the first, a path $p_{1i} \in P_{1i}(G^-_{A^*})$, $i \in V\backslash\{n\}$, belongs to ST when

$$\max_{u\in U_k}\{z^u_{G^-_{A^*}}(p_{1i}) + LB^{s_u}_{in}(G^-_{A^*})\} < UB. \tag{4.13}$$

For the second rule, to add an arc (i,j) to $p_{1i} \in ST \cap P_{1i}(G^-_{A^*})$, $i \in V\backslash\{n\}$, condition

$$\max_{u\in U_k}\{z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^-_{A^*}) + LB^{s_u}_{jn}(G^-_{A^*})\} < UB. \tag{4.14}$$

must hold. As before, this rule allows to set the sub-paths $\tilde{p}_{1i}$, $i \in V\backslash\{n\}$, in ST, with arcs that were not removed from G. Then, the initial X is set to

$$X = \{\langle 1\rangle\} \cup \{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in A(\tilde{p}) \cap (A\backslash A^*), j \neq n, \text{ satisfy (4.14)}\}.$$

Afterwards, the arcs candidate to extend each $p_{1i} \in ST \cap P_{1i}(G^-_{A^*})$, $Ad_{G^-_{A^*}}(p_{1i}\,|\,\tilde{p})$, $i \in V\backslash\{n\}$, are determined as in Algorithms 8 and 9, without considering the arcs of A^*, i.e.

$$Ad_{G^-_{A^*}}(p_{1i}\,|\,\tilde{p}) = \begin{cases} \left\{(i,j) \in A\backslash(A^* \cup A(\tilde{p})) : j \notin V(p_{1i})\right\} & \text{if} \quad p_{1i} = \tilde{p}_{1i} \\ \\ \left\{(i,j) \in A\backslash A^* : j \notin V(p_{1i})\right\} & \text{if} \quad p_{1i} \neq \tilde{p}_{1i} \end{cases}$$

Rule (4.14) is required for adding $(i,j) \in Ad_{G^-_{A^*}}(p_{1i}\,|\,\tilde{p})$, with $j \neq n$, to p_{1i}. When $j = n$, a third extension rule is applied, depending on the existence of an arc in $A(\tilde{p}) \cap A^*$.

If $A(\tilde{p}) \cap A^* = \emptyset$, Proposition 4.8 states that a simple path in set $\widehat{P}_{1n}(G^-_{A^*})$ with the least robustness cost in $G^-_{A^*}$ is the robust shortest path. Aiming at searching for this path, the produced $(1,n)$-path p_{1n} must satisfy $RD^{s_u}_G(p_{1n}) \geq RC_G(\tilde{p})$, for some $u \in \widehat{U}^A_{A\backslash A^*}$, and have a robustness cost in $G^-_{A^*}$, $RC_{G^-_{A^*}}(p_{1n})$, denoted by $RCaux$, that improves UB. Both conditions can be expressed in terms of the labels in $G^-_{A^*}$. In fact, by point 1. of Corollary 4.7,

$$RD^{s_u}_G(p_{1n}) = RD^{s_u}_{G^-_{A^*}}(p_{1n}) - LB^{s_u}_{1n}(G) + LB^{s_u}_{1n}(G^-_{A^*}),$$

and, by definition of robustness cost,

$$RCaux = \max_{u \in U_k} RD^{s_u}_{G^-_{A^*}}(p_{1n}),$$

with

$$RD^{s_u}_{G^-_{A^*}}(p_{1n}) = z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{in}(G^-_{A^*}),\ u \in U_k.$$

Therefore,

$$\begin{cases} RD^{s_u}_{G}(p_{1n}) = z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{in}(G^-_{A^*}) - LB^{s_u}_{1n}(G) + LB^{s_u}_{1n}(G^-_{A^*}) \geq RC_G(\tilde{p}),\ \text{for some } u \in \widehat{U}^A_{A \setminus A^*} \\ RCaux = \max_{u \in U_k}\{z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{in}(G^-_{A^*})\} < UB \end{cases} \tag{4.15}$$

must be fulfilled, in order to p_{1n} be potentially optimal in $G^-_{A^*}$.

If $A(\tilde{p}) \cap A^* \neq \emptyset$, the search for a robust shortest path of $G^-_{A^*}$ is extended to the simple paths of the set $P_{1n}(G^-_{A^*})$, with the least robustness cost in $G^-_{A^*}$. This means there is only need to check the second condition of (4.15) to arc (i,n) extend p_{1i}, $i \in V \setminus \{n\}$.

In either case, if the conditions associated with the third extension rule hold, *sol* and UB are updated to p_{1n} and $RCaux$, respectively. Moreover, only the paths of list X satisfying (4.13) are considered for the next iteration.

The pseudo-code of the procedure described above is presented in Algorithm 10.

Computational time complexity order Let m^* be the number of arcs removed from G. Then, $\underline{n}$ and $\underline{m} = m - m^*$ denote the number of nodes and arcs in $G^-_{A^*}$, respectively, with $2 \leq \underline{n} \leq n$ and $\underline{n} - 1 < \underline{m} < m$, so that $G^-_{A^*}$ is connected. Let W^-_A be the maximum number of paths generated in $ST \cap P_{1i}(G^-_{A^*})$, $i \in V$. Like for Algorithms 8 and 9, Algorithm 10 has two major parts, the first concerned with the procedures that precede the computation of ST, and the second devoted to the latter task in case it is required.

Similarly to previous algorithms, in the first stage, the trees $\mathcal{T}^{s_u}_n(G^-_{A^*})$ and the costs $LB^{s_u}_{in}(G^-_{A^*})$, $i \in V$, $u \in U_k$, are computed in time of $\mathcal{O}(k\underline{m})$ for acyclic networks and of $\mathcal{O}(k(\underline{m} + \underline{n}\log\underline{n}))$ for general networks. The determination of $A(\tilde{p}) \cap A^*$ may require $\mathcal{O}(n + m^*)$ operations, by using hash sets [8]. If $A(\tilde{p}) \cap A^* = \emptyset$, initializing UB implies computing set $\widehat{U}^A_{A \setminus A^*}$ in $\mathcal{O}(k)$ time and $RC_{G^-_{A^*}}(\tilde{p})$ in $\mathcal{O}(k\underline{n})$ time, as $\tilde{p} \in P_{1n}(G^-_{A^*})$. If $A(\tilde{p}) \cap A^* \neq \emptyset$, initializing UB implies, in the worst case, the determination of the k robustness costs $RC_{G^-_{A^*}}(p^{1,s_u}(G^-_{A^*}))$, $u \in U_k$, in $\mathcal{O}(k^2\underline{n})$ time. Therefore, in the worst case, the total time complexity invested on the first stage is of $O^a_1 = \mathcal{O}(\max\{k\underline{m}, m^*\} + \max\{k^2\underline{n}, n\})$ for acyclic networks and of $O^c_1 = \mathcal{O}(\max\{k\underline{m}, m^*\} + \max\{k^2\underline{n} + k\underline{n}\log\underline{n}, n\})$ for general networks.

Algorithm 10: Finding the robust shortest path in $G^-_{A^*}$, given $\tilde{p}$, $RC_G(\tilde{p})$ and $LB^{s_u}_{1n}(G)$, $u \in U_k$

for $u \in U_k$ **do** Compute $\mathcal{T}^{s_u}_n(G^-_{A^*})$ and $LB^{s_u}_{in}(G^-_{A^*})$, $i \in V$;
$\widehat{U}^A_{A\setminus A^*} \leftarrow \emptyset$; $UB \leftarrow 0$;
if $A(\tilde{p}) \cap A^* = \emptyset$ **then**
  $sol \leftarrow \tilde{p}$;
  for $u \in U_k$ **do**
    if $LB^{s_u}_{1n}(G) < LB^{s_u}_{1n}(G^-_{A^*})$ **then** $\widehat{U}^A_{A\setminus A^*} \leftarrow \widehat{U}^A_{A\setminus A^*} \cup \{u\}$;
  if $\widehat{U}^A_{A\setminus A^*} \neq \emptyset$ **then** $RC_{G^-_{A^*}}(\tilde{p}) \leftarrow \max_{u\in U_k} RD^{s_u}_{G^-_{A^*}}(\tilde{p})$; $UB \leftarrow RC_{G^-_{A^*}}(\tilde{p})$;
else
  $UB \leftarrow \min\{RC_{G^-_{A^*}}(p^{1,s_u}(G^-_{A^*})) : u \in U_k\}$; $sol \leftarrow p^{1,s_{u'}}(G^-_{A^*})$ such that $RC_{G^-_{A^*}}(p^{1,s_{u'}}(G^-_{A^*})) = UB$, $u' \in U_k$;
if $UB \neq 0$ ***and*** $(\widehat{U}^A_{A\setminus A^*} \neq \emptyset$ ***or*** $A(\tilde{p}) \cap A^* \neq \emptyset)$ **then**
  $X \leftarrow \{\langle 1\rangle\}$;
  for $u \in U_k$ **do** $z^u_{G^-_{A^*}}(\langle 1\rangle) \leftarrow -LB^{s_u}_{1n}(G^-_{A^*})$;
  for $(i,j) \in A(\tilde{p}) \cap (A\setminus A^*)$ ***and*** $j \neq n$ **do**
    if $\max_{u\in U_k}\{z^u_{G^-_{A^*}}(\tilde{p}_{1i}) + c^{s_u}_{ij}(G^-_{A^*}) + LB^{s_u}_{jn}(G^-_{A^*})\} < UB$ **then**
      $X \leftarrow X \cup \{\tilde{p}_{1j}\}$;
      for $u \in U_k$ **do** $z^u_{G^-_{A^*}}(\tilde{p}_{1j}) \leftarrow z^u_{G^-_{A^*}}(\tilde{p}_{1i}) + c^{s_u}_{ij}(G^-_{A^*})$;
    else break;
  while $X \neq \emptyset$ **do**
    $p_{1i} \leftarrow$ first path in X; $X \leftarrow X - \{p_{1i}\}$; Compute $Ad_{G^-_{A^*}}(p_{1i}\,|\,\tilde{p})$;
    for $(i,j) \in Ad_{G^-_{A^*}}(p_{1i}\,|\,\tilde{p})$ **do**
      $p_{1j} \leftarrow p_{1i} \diamond \langle i,j\rangle$;
      if $j = n$ **then**
        if $A(\tilde{p}) \cap A^* = \emptyset$ **then**
          for $u \in \widehat{U}^A_{A\setminus A^*}$ **do**
            $RD^{s_u}_G(p_{1j}) \leftarrow z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^-_{A^*}) - LB^{s_u}_{1n}(G) + LB^{s_u}_{1n}(G^-_{A^*})$;
            if $RD^{s_u}_G(p_{1j}) \geq RC_G(\tilde{p})$ **then**
              $RCaux \leftarrow \max_{u\in U_k}\{z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^-_{A^*})\}$; **break**;
        else $RCaux \leftarrow \max_{u\in U_k}\{z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^-_{A^*})\}$;
        if $RCaux < UB$ **then**
          $UB \leftarrow RCaux$; $sol \leftarrow p_{1j}$;
          for $p_{1i'} \in X$ **do**
            for $u \in U_k$ **do**
              if $z^u_{G^-_{A^*}}(p_{1i'}) + LB^{s_u}_{i'n}(G^-_{A^*}) \geq UB$ **then**
                $X \leftarrow X - \{p_{1i'}\}$; **break**;
      else
        if $\max_{u\in U_k}\{z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^-_{A^*}) + LB^{s_u}_{jn}(G^-_{A^*})\} < UB$ **then**
          $X \leftarrow X \cup \{p_{1j}\}$;
          for $u \in U_k$ **do** $z^u_{G^-_{A^*}}(p_{1j}) \leftarrow z^u_{G^-_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^-_{A^*})$;
return sol;

The procedure for the second stage is similar to the performed for developing ST in Algorithm 9. They differ on the number of nodes, arcs and scenarios, which are now $\underline{n}$, $\underline{m}$ and k, respectively, and on the third extension rule. However, like for Algorithm 9, the complexity bound of this rule depends only on the number of scenarios and requires $\mathcal{O}(k)$ time. With a similar reasoning, the second stage of Algorithm 10 performs in $O_2 = \mathcal{O}(k\underline{n}^3(W_A^-)^2)$ time.

In conclusion, as $\log \underline{n} \ll \underline{n}$ and $\underline{m} < \underline{n}^2$, the total time complexity for Algorithm 10 is $\mathcal{O}(m^* + \max\{k^2\underline{n} + k\underline{n}^3(W_A^-)^2, n\})$ for any type of network. It can be noted that besides the number of arcs and nodes in $G^-_{A^*}$, the time complexity depends also on the number of arcs and nodes deleted from G.

Examples In the following, three networks obtained from G_5 – Figure 4.1, after deleting some of its arcs are presented. The robust shortest path in G_5 is $\tilde{p} = \langle 1, 4, 6, 7\rangle$. In the first two examples, the deleted arcs do not belong to $\tilde{p}$, whereas, in the third example, one arc of $\tilde{p}$ is removed. Initially, $A^* = \{(6,5)\}$, which results in $\widehat{U}^A_{A\setminus A^*} = \emptyset$. Then, $A^* = \{(6,5),(5,3)\}$, which makes that $\widehat{U}^A_{A\setminus A^*} \neq \emptyset$, and, finally, $A^* = \{(6,5),(5,3),(4,6)\}$, where $(4,6)$ is the only deleted arc that belongs to $\tilde{p}$.

Case 1 Let $(G_5)^-_{(6,5)}$ be the network in Figure 4.9.

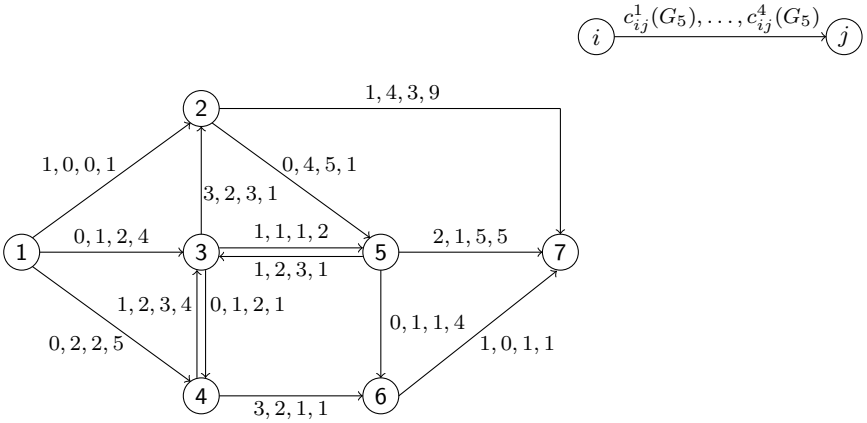


Figure 4.9: Network $(G_5)^-_{(6,5)}$

Recalling Figures 4.3 and 4.7.(a),(b), $(6,5) \notin A(p^{1,u}(G_5))$, for any $u \in U_4$, which means that

$$LB^u_{17}((G_5)^-_{(6,5)}) = LB^u_{17}(G_5)\,,\ u \in U_4.$$

Since $A(\tilde{p}) \cap A^* = \emptyset$, the calculation of set $\widehat{U}^A_{A\setminus\{(6,5)\}}$ is demanded. Then,

$$\widehat{U}^A_{A\setminus\{(6,5)\}} = \emptyset.$$

Therefore, attending to 1. of Proposition 4.8, $\tilde{p}$ is returned by Algorithm 10 as the robust shortest path of $(G_5)^-_{(6,5)}$, and $RC_{(G_5)^-_{(6,5)}}(\tilde{p}) = RC_{G_5}(\tilde{p}) = 2$.

Case 2 Let now $(G_5)^-_{A^*}$, with $A^* = \{(6,5),(5,3)\}$, be the network in Figure 4.10.

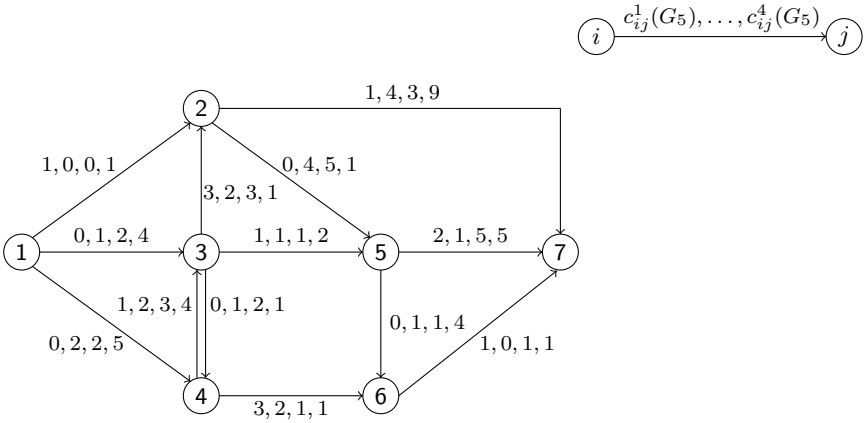


Figure 4.10: Network $(G_5)^-_{A^*}$, with $A^* = \{(6,5),(5,3)\}$

According with Figures 4.3 and 4.7, arcs $(6,5)$ and $(5,3)$ do not belong to any tree $\mathcal{T}^u_7(G_5)$, $u \in U_3$, which means that these trees remain unchanged in $(G_5)^-_{A^*}$, i.e.

$$\mathcal{T}^u_7(G^-_{A^*}) = \mathcal{T}^u_7(G_5)\,,\ u \in U_3.$$

Since the arc $(5,3)$ in $\mathcal{T}^4_7(G_5)$ was excluded, the tree $\mathcal{T}^4_7((G_5)^-_{A^*})$ is now determined – Figure 4.11.

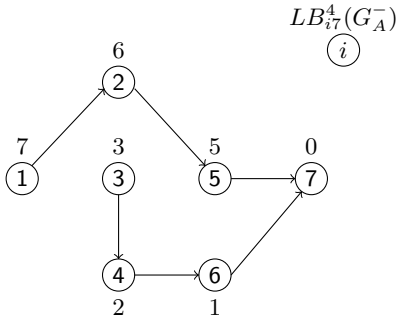


Figure 4.11: Tree $\mathcal{T}^4_7((G_5)^-_{A^*})$, with $A^* = \{(6,5),(5,3)\}$

As $A(\tilde{p}) \cap A^* = \emptyset$, Algorithm 10 determines set $\widehat{U}^A_{A\setminus A^*}$. Then,

$$\widehat{U}^A_{A\setminus A^*} = \{u \in U_4 : LB^u_{17}(G_5) < LB^u_{17}((G_5)^-_{A^*})\} = \{4\}.$$

Because $\widehat{U}^{A}_{A\setminus A^*} \neq \emptyset$, UB and sol are initialized with

$$UB = RC_{(G_5)^-_{A^*}}(\tilde{p}) = \max_{u\in U_4} RD^u_{(G_5)^-_{A^*}}(\tilde{p}) = 2 \quad \text{and} \quad sol = \tilde{p} = \langle 1,4,6,7\rangle.$$

Since $UB \neq 0$, ST is constructed, starting with the sub-paths of $\tilde{p}$ in

$$X = \big\{\langle 1\rangle\big\} \cup \big\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in \{(1,4),(4,6)\} \text{ satisfy (4.14)}\big\}.$$

From the label

$$z_{(G_5)^-_{A^*}}(\langle 1\rangle) = (-2,-3,-3,-7),$$

one concludes that arc $(1,4)$ can extend $\langle 1\rangle$, because

$$\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1\rangle) + c^u_{14}((G_5)^-_{A^*}) + LB^u_{47}((G_5)^-_{A^*})\} = 1 < UB,$$

and, then, the label

$$z_{(G_5)^-_{A^*}}(\langle 1,4\rangle) = (-2,-1,-1,-2)$$

is created. However, path $\langle 1,4,6\rangle$ does not belong to ST, because

$$\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1,4\rangle) + c^u_{46}((G_5)^-_{A^*}) + LB^u_{67}((G_5)^-_{A^*})\} = 2 \geq UB,$$

and, therefore, X is initialized with

$$X = \{\langle 1\rangle, \langle 1,4\rangle\}.$$

Figure 4.12 shows the ST obtained by means of Algorithm 10 for $(G_5)^-_{A^*}$.

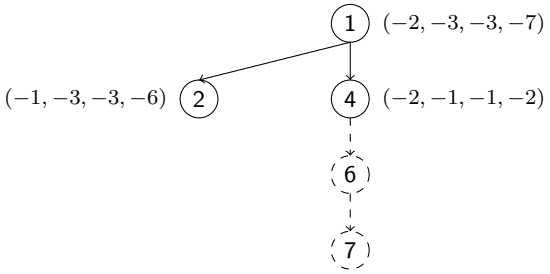


Figure 4.12: ST produced by Algorithm 10 for $(G_5)^-_{A^*}$, with $A^* = \{(6,5),(5,3)\}$

The arcs candidate for extending $\langle 1\rangle$ in ST are those in

$$Ad_{(G_5)^-_{A^*}}(\langle 1\rangle \,|\, \tilde{p}) = \{(1,j) \in A\setminus(A^* \cup A(\tilde{p}))\} = \{(1,2),(1,3)\}.$$

Only arc $(1,2)$ can be used, since

$$\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1\rangle) + c^u_{12}((G_5)^-_{A^*}) + LB^u_{27}((G_5)^-_{A^*})\} = 1 < UB$$

while, for arc $(1,3)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1\rangle)+c^u_{13}((G_5)^-_{A^*})+LB^u_{37}((G_5)^-_{A^*})\}=2\geq UB.$$

When adding arc $(1,2)$, the label

$$z_{(G_5)^-_{A^*}}(\langle 1,2\rangle)=(-1,-3,-3,-6)$$

is determined, and X is updated to

$$X=\{\langle 1,4\rangle,\langle 1,2\rangle\}.$$

Next, $\langle 1,4\rangle$ is chosen to be extended, with

$$Ad_{(G_5)^-_{A^*}}(\langle 1,4\rangle\,|\,\tilde{p})=\{(4,j)\in A\backslash(A^*\cup A(\tilde{p})):j\neq 1\}=\{(4,3)\}.$$

However, arc $(4,3)$ cannot be used, as

$$\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1,4\rangle)+c^u_{43}((G_5)^-_{A^*})+LB^u_{37}((G_5)^-_{A^*})\}=5\geq UB.$$

For the path $\langle 1,2\rangle$,

$$Ad_{(G_5)^-_{A^*}}(\langle 1,2\rangle\,|\,\tilde{p})=\{(2,j)\in A\backslash A^*:j\neq 1\}=\{(2,5),(2,7)\},$$

and none of these arcs is included in ST, given that, for arc $(2,5)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1,2\rangle)+c^u_{25}((G_5)^-_{A^*})+LB^u_{57}((G_5)^-_{A^*})\}=4\geq UB$$

and, for arc $(2,7)$,

$$\begin{cases} RD^4_{G_5}(\langle 1,2,7\rangle)=z^4_{(G_5)^-_{A^*}}(\langle 1,2\rangle)+c^4_{27}((G_5)^-_{A^*})-LB^4_{17}(G_5)+LB^4_{17}((G_5)^-_{A^*})=4\geq RC_{G_5}(\tilde{p}) \\ RCaux=\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1,2\rangle)+c^u_{27}((G_5)^-_{A^*})\}=3\geq UB \end{cases}$$

Then, there are no other paths in X to extend. Consequently, Algorithm 10 halts, outputting $\tilde{p}$ as the robust shortest path in $(G_5)^-_{A^*}$, with $RC_{(G_5)^-_{A^*}}(\tilde{p})=RC_{G_5}(\tilde{p})=2$.

Case 3 Let now $(G_5)^-_{A^*}$, with $A^*=\{(6,5),(5,3),(4,6)\}$, be the network in Figure 4.13. Since the tree $\mathcal{T}^1_7(G_5)$ in Figure 4.3.(a) does not contain any of the deleted arcs from G_5, then

$$\mathcal{T}^1_7((G_5)^-_{A^*})=\mathcal{T}^1_7(G_5).$$

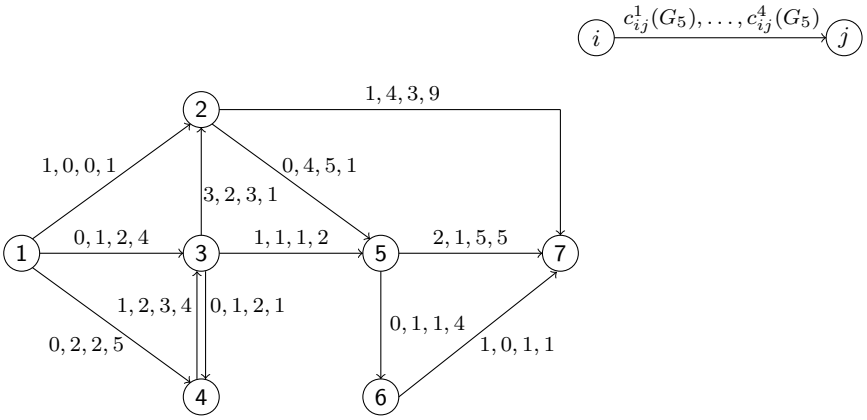


Figure 4.13: Network $(G_5)^-_{A^*}$, with $A^* = \{(6,5),(5,3),(4,6)\}$

Figures 4.7 and 4.3 show that all the trees in G_5 for the remaining scenarios, contain at least one of the arcs excluded from G_5. Hence, the homologous trees in $(G_5)^-_{A^*}$ are determined – Figure 4.14.

Because arc $(4,6)$ of $\tilde{p}$ is removed from G_5, Algorithm 10 considers the least robustness cost in $(G_5)^-_{A^*}$ for the paths $p^{1,s_u}((G_5)^-_{A^*})$, $u \in U_4$, to set the initial UB. As

$$RC_{(G_5)^-_{A^*}}(p^{1,u}((G_5)^-_{A^*})) = RC_{(G_5)^-_{A^*}}(\langle 1,2,7\rangle) = 3, \quad u = 1,3,$$

$$RC_{(G_5)^-_{A^*}}(p^{1,2}((G_5)^-_{A^*})) = RC_{(G_5)^-_{A^*}}(\langle 1,3,5,7\rangle) = 5$$

and

$$RC_{(G_5)^-_{A^*}}(p^{1,4}((G_5)^-_{A^*})) = RC_{(G_5)^-_{A^*}}(\langle 1,2,5,7\rangle) = 7,$$

it follows that

$$UB = \min\{RC_{(G_5)^-_{A^*}}(p^{1,u}((G_5)^-_{A^*})) : u \in U_4\} = 3 \quad \text{and} \quad sol = \langle 1,2,7\rangle.$$

Because $UB \neq 0$ and $(4,6) \in A^*$, ST starts to consider the sub-paths of $\tilde{p}$ in

$$X = \big\{\langle 1\rangle\big\} \cup \big\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in \{(1,4)\} \text{ satisfy (4.14)}\big\}.$$

From the label,

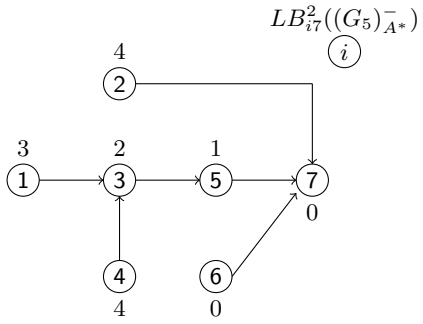
$$z_{(G_5)^-_{A^*}}(\langle 1\rangle) = (-2,-3,-3,-7),$$

one concludes that path $\langle 1,4\rangle$ does not belong to ST, as

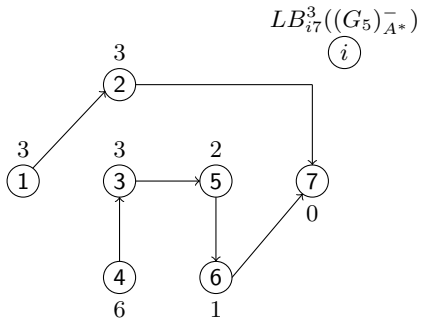
$$\max_{u\in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1\rangle) + c^u_{14}((G_5)^-_{A^*}) + LB^u_{47}((G_5)^-_{A^*})\} = 9 \geq UB.$$

Hence, ST starts with

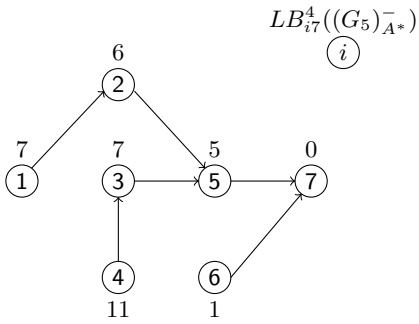
$$X = \{\langle 1\rangle\}.$$



(a) $\mathcal{T}_7^2((G_5)^-_{A^*})$



(b) $\mathcal{T}_7^3((G_5)^-_{A^*})$



(c) $\mathcal{T}_7^4((G_5)^-_{A^*})$

Figure 4.14: Shortest path trees rooted at node 7 for scenarios 2, 3 and 4 in $(G_5)^-_{A^*}$, with $A^* = \{(6,5),(5,3),(4,6)\}$

The ST obtained by Algorithm 10 for $(G_5)^-_{A^*}$ is shown in Figure 4.15. The arcs of

$$Ad_{(G_5)^-_{A^*}}(\langle 1\rangle \,|\, \tilde{p}) = \{(1,j) \in A \backslash (A^* \cup A(\tilde{p}))\} = \{(1,2),(1,3)\}$$

are analyzed for extension from node 1. Only arc $(1,2)$ is used, since

$$\max_{u \in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1\rangle) + c^u_{12}((G_5)^-_{A^*}) + LB^u_{27}((G_5)^-_{A^*})\} = 1 < UB,$$

while, for arc $(1,3)$,

$$\max_{u \in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1\rangle) + c^u_{13}((G_5)^-_{A^*}) + LB^u_{37}((G_5)^-_{A^*})\} = 4 \geq UB.$$

When adding arc $(1,2)$ to $\langle 1\rangle$, the associated label is of

$$z_{(G_5)^-_{A^*}}(\langle 1,2\rangle) = (-1,-3,-3,-6).$$

Then, X is updated to

$$X = \{\langle 1,2\rangle\}.$$

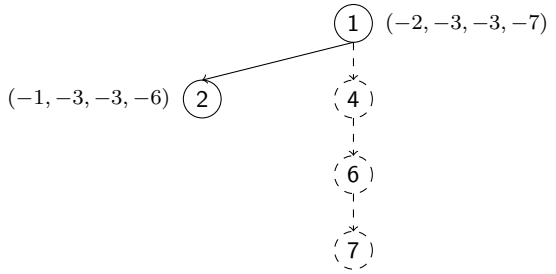


Figure 4.15: ST produced by Algorithm 10 for $G^-_{A^*}$, with $A^* = \{(6,5),(5,3),(4,6)\}$

There are now two possibilities for extending $\langle 1,2\rangle$,

$$Ad_{(G_5)^-_{A^*}}(\langle 1,2\rangle \,|\, \tilde{p}) = \{(2,j) \in A \backslash A^* : j \neq 1\} = \{(2,5),(2,7)\}.$$

However, these arcs cannot be included in ST, since, for arc $(2,5)$,

$$\max_{u \in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1,2\rangle) + c^u_{25}((G_5)^-_{A^*}) + LB^u_{57}((G_5)^-_{A^*})\} = 4 \geq UB$$

and, for arc $(2,7)$,

$$RCaux = \max_{u \in U_4}\{z^u_{(G_5)^-_{A^*}}(\langle 1,2\rangle) + c^u_{27}((G_5)^-_{A^*})\} = 3 \geq UB,$$

which means that the second condition of (4.15) is not satisfied. Then, Algorithm 10 stops, returning $\langle 1,2,7\rangle$ as the robust shortest path in $(G_5)^-_{A^*}$, with $RC_{(G_5)^-_{A^*}}(\langle 1,2,7\rangle) = 3$.

In the following, it is addressed the reoptimization procedure when the set of arcs in G is extended.

4.3.2 Addition of arcs

Let $A^* \subseteq V \times V$ be a finite set of arcs added to A in G. Then, network $G^+_{A^*}$ has set of nodes V, set of arcs $A \cup A^*$ and set of $(1,n)$-paths $P_{1n}(G^+_{A^*})$, which contains $P_{1n}(G)$. Like in the previous subsection, Lemma 4.6 is adapted, but now for the networks $G'_A = G$ and $G''_A = G^+_{A^*}$.

Corollary 4.9. *For any $p \in P_{1n}(G)$,*

1. $RD^{s_u}_G(p) = RD^{s_u}_{G^+_{A^*}}(p) + LB^{s_u}_{1n}(G^+_{A^*}) - LB^{s_u}_{1n}(G)$, *for any $u \in U_k$. In particular,* $RD^{s_u}_G(p) = RD^{s_u}_{G^+_{A^*}}(p)$, *for any $u \in U_k \backslash \widehat{U}^{A \cup A^*}_A$.*

2. $RC_G(p) \leq RC_{G^+_{A^*}}(p)$. *In particular, if $\widehat{U}^{A \cup A^*}_A = \emptyset$, then $RC_G(p) = RC_{G^+_{A^*}}(p)$.*

According to Chapter 3, the arcs of A^* identified as robust 0-persistent in $G^+_{A^*}$ can be skipped to determine a robust shortest path in $G^+_{A^*}$, which simplifies the search method. The following result derives a condition for detecting those arcs and also conditions under which $\tilde{p}$ is a robust shortest path in $G^+_{A^*}$, taking into account set $\widehat{U}_A^{A\cup A^*}$. Otherwise, an optimal solution can be searched in a subset of paths in $P_{1n}(G^+_{A^*})$, which do not contain any of the identified robust 0-persistent arcs.

Proposition 4.10. *Let*

$$\hat{A} = \{(i,j) \in A^* : RD^{s_u}_{G^+_{A^*}}(p^{1,s_u}_{1i}(G^+_{A^*}) \diamond \langle i,j\rangle \diamond p^{1,s_u}_{jn}(G^+_{A^*})) > RC_{G^+_{A^*}}(\tilde{p}), \text{ for some } u \in U_k\}$$

be the set of arcs added to A in G satisfying the condition above, and

$$\bar{P}_{1n}(G^+_{A^*}) = \{p \in P_{1n}(G^+_{A^*}) : A(p) \cap \hat{A} = \emptyset\},$$

represent the set of $(1,n)$-paths in $G^+_{A^*}$ which do not contain any arc in $\hat{A}$. Then,

1. *Any arc $(i,j) \in \hat{A}$ is robust 0-persistent in $G^+_{A^*}$ and*

$$RC_{G^+_{A^*}}(p) > RC_{G^+_{A^*}}(\tilde{p}), \text{ for any } p \in P_{1n}(G^+_{A^*}) \backslash \bar{P}_{1n}(G^+_{A^*}).$$

2. *If $\widehat{U}_A^{A\cup A^*} = \emptyset$, then $RC_{G^+_{A^*}}(\tilde{p}) = RC_G(\tilde{p})$. Moreover, $\tilde{p}$ is a robust shortest path in $G^+_{A^*}$, if and only if*

$$RC_{G^+_{A^*}}(p) \geq RC_{G^+_{A^*}}(\tilde{p}), \text{ for any } p \in \bar{P}_{1n}(G^+_{A^*}) \backslash P_{1n}(G).$$

3. *If $\widehat{U}_A^{A\cup A^*} \neq \emptyset$, let $\widehat{P}_{1n}(G^+_{A^*}) \subseteq \bar{P}_{1n}(G^+_{A^*})$ be given by*

$$\widehat{P}_{1n}(G^+_{A^*}) = \{p \in \bar{P}_{1n}(G^+_{A^*}) : \max_{u \in U_k \backslash \widehat{U}_A^{A\cup A^*}} RD^{s_u}_{G^+_{A^*}}(p) < RC_{G^+_{A^*}}(\tilde{p})\}.$$

   (a) *If $p \in P_{1n}(G^+_{A^*})$ satisfies $RC_{G^+_{A^*}}(p) < RC_{G^+_{A^*}}(\tilde{p})$, then $p \in \widehat{P}_{1n}(G^+_{A^*})$.*

   (b) *$\tilde{p}$ is a robust shortest path in $G^+_{A^*}$ if and only if $RC_{G^+_{A^*}}(p) \geq RC_{G^+_{A^*}}(\tilde{p})$, for any $p \in \widehat{P}_{1n}(G^+_{A^*})$.*

Proof.

1. The proof is similar to the presented for Proposition 3.2. For completeness, some of its steps are outlined in the following.

   Let $(i,j) \in \hat{A}$ and $p \in P_{1n}(G^+_{A^*}) \backslash \bar{P}_{1n}(G^+_{A^*})$, such that $(i,j) \in A(p)$. Because $p^{1,s_u}_{1i}(G^+_{A^*})$ and $p^{1,s_u}_{jn}(G^+_{A^*})$ are the shortest $(1,i)$-path and the shortest (j,n)-path for scenario s_u, $u \in U_k$, in $G^+_{A^*}$, respectively,

$$RD^{s_u}_{G^+_{A^*}}(p) \geq RD^{s_u}_{G^+_{A^*}}(p^{1,s_u}_{1i}(G^+_{A^*}) \diamond \langle i,j\rangle \diamond p^{1,s_u}_{jn}(G^+_{A^*})).$$

Since $(i,j) \in \hat{A}$,

$$RD^{s_u}_{G^+_{A^*}}(p) > RC_{G^+_{A^*}}(\tilde{p}), \text{ for some } u \in U_k,$$

according to the definition of set $\hat{A}$. By definition of robustness cost,

$$RC_{G^+_{A^*}}(p) = \max_{u \in U_k} RD^{s_u}_{G^+_{A^*}}(p) > RC_{G^+_{A^*}}(\tilde{p}).$$

Hence, any path $p \in P_{1n}(G^+_{A^*})$ containing (i,j) cannot be a robust shortest path in $G^+_{A^*}$. Moreover, any path $p \in P_{1n}(G^+_{A^*}) \backslash \bar{P}_{1n}(G^+_{A^*})$ satisfies the previous condition.

2. Assume that $\widehat{U}^{A \cup A^*}_A = \emptyset$. Then, by point 2. of Corollary 4.9 for $\tilde{p}$,

$$RC_{G^+_{A^*}}(\tilde{p}) = RC_G(\tilde{p}).$$

Before proving the remain part of the result, it should be noted that, when $\hat{A} = A^*$, $\bar{P}_{1n}(G^+_{A^*}) \backslash P_{1n}(G) = \emptyset$. This fact does not affect the validity of the following reasoning.

Let $\tilde{p}$ be a robust shortest path of $G^+_{A^*}$, then, by definition,

$$RC_{G^+_{A^*}}(p) \geq RC_{G^+_{A^*}}(\tilde{p}), \text{ for any } p \in P_{1n}(G^+_{A^*}), \tag{4.16}$$

and, in particular,

$$RC_{G^+_{A^*}}(p) \geq RC_{G^+_{A^*}}(\tilde{p}), \text{ for any } p \in \bar{P}_{1n}(G^+_{A^*}) \backslash P_{1n}(G).$$

Conversely, assume this condition holds. Since, by hypothesis, $\tilde{p}$ is a robust shortest path of G,

$$RC_G(p) \geq RC_G(\tilde{p}), \text{ for any } p \in P_{1n}(G).$$

Given that $\widehat{U}^{A \cup A^*}_A = \emptyset$, point 2. of Corollary 4.9 for p and $\tilde{p}$ can rewrite the previous condition as

$$RC_{G^+_{A^*}}(p) \geq RC_{G^+_{A^*}}(\tilde{p}), \text{ for any } p \in P_{1n}(G).$$

Consequently, by the assumption, the last condition and point 1., (4.16) follows, i.e. $\tilde{p}$ is a robust shortest path in $G^+_{A^*}$.

3. Assume that $\widehat{U}^{A \cup A^*}_A \neq \emptyset$.

   (a) Let $p \in P_{1n}(G^+_{A^*})$ be such that $RC_{G^+_{A^*}}(p) < RC_{G^+_{A^*}}(\tilde{p})$. Then, by point 1., one must have $p \in \bar{P}_{1n}(G^+_{A^*})$. Moreover, by definition of robustness cost,

$$\max_{u \in U_k \backslash \widehat{U}^{A \cup A^*}_A} RD^{s_u}_{G^+_{A^*}}(p) \leq \max_{u \in U_k} RD^{s_u}_{G^+_{A^*}}(p) = RC_{G^+_{A^*}}(p) < RC_{G^+_{A^*}}(\tilde{p}).$$

   Hence, $p \in \widehat{P}_{1n}(G^+_{A^*})$.

(b) The result is derived as in points 2. of Proposition 4.3 and 2.(b) of Propositions 4.5 and 4.8, by considering the network $G^+_{A^*}$ and the set $\widehat{P}_{1n}(G^+_{A^*})$.

□

From Proposition 4.10, the determination of sets $\hat{A}$ and $\widehat{U}_A^{A\cup A^*}$ allows to restrict the search for simple robust shortest paths in $P_{1n}(G^+_{A^*})$. In fact, knowing set $\hat{A}$, allows to discard the robust 0-persistent arcs of A^*, and, if set $\widehat{U}_A^{A\cup A^*}$ is known, a subset of $P_{1n}(G^+_{A^*})$ containing a robust shortest path in $G^+_{A^*}$ is determined. Specifically, when $\widehat{U}_A^{A\cup A^*} = \emptyset$, if $\hat{A} = A^*$, $\tilde{p}$ is returned as the robust shortest path of $G^+_{A^*}$, because $\bar{P}_{1n}(G^+_{A^*})\backslash P_{1n}(G) = \emptyset$. Otherwise, the search is focused on the simple paths in the latter set, with the least robustness cost in $G^+_{A^*}$. When $\widehat{U}_A^{A\cup A^*} \neq \emptyset$, the simple paths of set $\widehat{P}_{1n}(G^+_{A^*})$ are analyzed instead.

The algorithm for reoptimizing the robust shortest path in $G^+_{A^*}$ starts by determining set $\widehat{U}_A^{A\cup A^*}$ and then, by initializing UB and sol with $RC_{G^+_{A^*}}(\tilde{p})$ and $\tilde{p}$, respectively. If $UB = 0$, then $\tilde{p}$ is returned as the robust shortest path of $G^+_{A^*}$, else, set $\hat{A}$ is calculated, by means of computing the trees $\mathcal{T}_1^{s_u}(G^+_{A^*})$, $u \in U_k$. Using the strategy in Chapter 3, in order to spare computational effort, for each $(i,j) \in A^*$, it is only necessary to compute those trees for the scenarios indices up to the smallest M_{ij}, $M_{ij} \leq k$, such that

$$RD^{s_{M_{ij}}}_{G^+_{A^*}}(p_{1i}^{1,s_{M_{ij}}}(G^+_{A^*}) \diamond \langle i,j\rangle \diamond p_{jn}^{1,s_{M_{ij}}}(G^+_{A^*})) > UB,$$

noticing that, at this stage, $UB = RC_{G^+_{A^*}}(\tilde{p})$.

After knowing sets $\widehat{U}_A^{A\cup A^*}$ and $\hat{A}$, ST is built when $\tilde{p}$ cannot be assured as the optimal solution, i.e. when $\widehat{U}_A^{A\cup A^*} = \emptyset$ and $\hat{A} \subsetneq A^*$, or when $\widehat{U}_A^{A\cup A^*} \neq \emptyset$. The labeling process and the first and second extension rules in ST are analogous to the applied in the previous subsection, which are adapted to $G^+_{A^*}$ in the following. The labeling starts with

$$z_{G^+_{A^*}}(\langle 1\rangle) = (-LB_{1n}^{s_1}(G^+_{A^*}), \ldots, -LB_{1n}^{s_k}(G^+_{A^*})).$$

The first extension rule considers that a path $p_{1i} \in P_{1i}(G^+_{A^*})$, $i \in V\backslash\{n\}$, can be extended in ST when

$$\max_{u\in U_k}\{z^u_{G^+_{A^*}}(p_{1i}) + LB_{in}^{s_u}(G^+_{A^*})\} < UB. \tag{4.17}$$

The second extension rule considers that any arc (i,j) extending $p_{1i} \in ST \cap P_{1i}(G^+_{A^*})$, $i \in V\backslash\{n\}$, must satisfy

$$\max_{u\in U_k}\{z^u_{G^+_{A^*}}(p_{1i}) + c_{ij}^{s_u}(G^+_{A^*}) + LB_{jn}^{s_u}(G^+_{A^*})\} < UB. \tag{4.18}$$

With this rule, the sub-paths $\tilde{p}_{1i}$, $i \in V\backslash\{n\}$, belonging to ST, are set in the initial list X, given by

$$X = \{\langle 1\rangle\} \cup \left\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in A(\tilde{p}),\ j \neq n, \text{ satisfy (4.18)}\right\}.$$

The subsequent extension of each $p_{1i} \in ST \cap P_{1i}(G^+_{A^*})$, $i \in V\backslash\{n\}$, considers the arcs in $Ad_{G^+_{A^*}}(p_{1i}\,|\,\tilde{p})$. This set is determined according with the principles followed by Algorithms 8 and 9, without considering the arcs of $\hat{A}$, i.e.

$$Ad_{G^+_{A^*}}(p_{1i}\,|\,\tilde{p}) = \begin{cases} \left\{(i,j) \in (A \cup A^*)\backslash(\hat{A} \cup A(\tilde{p})) : j \notin V(p_{1i})\right\} & \text{if} \quad p_{1i} = \tilde{p}_{1i} \\ \\ \left\{(i,j) \in (A \cup A^*)\backslash\hat{A} : j \notin V(p_{1i})\right\} & \text{if} \quad p_{1i} \neq \tilde{p}_{1i} \end{cases}$$

A path $p_{1i} \in ST \cap P_{1i}(G^+_{A^*})$, $i \in V\backslash\{n\}$, is extended by an arc $(i,j) \in Ad_{G^+_{A^*}}(p_{1i}\,|\,\tilde{p})$, with $j \neq n$, if rule (4.18) is satisfied. When, $j = n$, a third extension rule is applied to the obtained $(1,n)$-path p_{1n}, when $\widehat{U}_A^{A\cup A^*} = \emptyset$ and $\hat{A} \subsetneq A^*$, or, when $\widehat{U}_A^{A\cup A^*} \neq \emptyset$. For the first case, a robust shortest path in $G^+_{A^*}$ is found among the paths in the set

$$\bar{P}_{1n}(G^+_{A^*})\backslash P_{1n}(G) = \{p \in P_{1n}(G^+_{A^*}) : A(p) \cap \left(A^*\backslash\hat{A}\right) \neq \emptyset\},$$

with a robustness cost in $G^+_{A^*}$, denoted by $RCaux$,

$$RCaux = \max_{u\in U_k} RD^{s_u}_{G^+_{A^*}}(p_{1n}),$$

which can be better than UB. Since in terms of the labels in $G^+_{A^*}$,

$$RD^{s_u}_{G^+_{A^*}}(p_{1n}) = z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{in}(G^+_{A^*})\,,\ u \in U_k,$$

the arc $(i,n) \in Ad_{G^+_{A^*}}(p_{1i}\,|\,\tilde{p})$ extends p_{1i} whenever

$$\begin{cases} A(p_{1n}) \cap \left(A^*\backslash\hat{A}\right) \neq \emptyset \\ \\ RCaux = \max_{u\in U_k}\{z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{in}(G^+_{A^*})\} < UB. \end{cases} \tag{4.19}$$

Instead, if $\widehat{U}_A^{A\cup A^*} \neq \emptyset$, the search for a robust shortest path in $G^+_{A^*}$ covers the simple paths in set $\widehat{P}_{1n}(G^+_{A^*})$, with a robustness cost in $G^+_{A^*}$, $RCaux$, that can improve UB. With a reasoning similar to the exposed in Subsection 4.2.2, the search set can be reduced whenever UB is updated, by considering it as

$$\widehat{P}^{(UB)}_{1n}(G^+_{A^*}) = \{p \in \bar{P}_{1n}(G^+_{A^*}) : \max_{u\in U_k\backslash\widehat{U}_A^{A\cup A^*}} RD^{s_u}_{G^+_{A^*}}(p) < UB\}.$$

Under these conditions, there is no need to analyze any path $p \in \bar{P}_{1n}(G^+_{A^*})$, such that $UB' \leq \max_{u \in U_k \setminus \widehat{U}_A^{A \cup A^*}} RD^{s_u}_{G^+_{A^*}}(p) < UB$, because in this case $RC_{G^+_{A^*}}(p) = \max_{u \in U_k} RD^{s_u}_{G^+_{A^*}}(p) \geq UB'$. Hence, when UB is updated, the search set is updated to $\widehat{P}^{(UB)}_{1n}(G^+_{A^*})$ as well. Therefore, arc $(i,n) \in Ad_{G^+_{A^*}}(p_{1i} \,|\, \tilde{p})$ extends p_{1i}, $i \in V \setminus \{n\}$, when the obtained $(1,n)$-path p_{1n} belongs to $\widehat{P}^{(UB)}_{1n}(G^+_{A^*})$, i.e. if

$$aux1 = \max_{u \in U_k \setminus \widehat{U}_A^{A \cup A^*}} RD^{s_u}_{G^+_{A^*}}(p) < UB,$$

and when $RC_{G^+_{A^*}}(p_{1n})$, i.e. $RCaux$, can improve UB. In fact, $RCaux$ can make use of $aux1$,

$$RCaux = \max\left\{aux1, \max_{u \in \widehat{U}_A^{A \cup A^*}} RD^{s_u}_{G^+_{A^*}}(p_{1n})\right\}.$$

The last two equalities can be expressed in terms of labels, using them to represent the robust deviations as above. Then, the two following conditions must hold

$$\begin{cases} aux1 = \max_{u \in U_k \setminus \widehat{U}_A^{A \cup A^*}} \{z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{in}(G^+_{A^*})\} < UB \\ \\ RCaux = \max\left\{aux1, \max_{u \in \widehat{U}_A^{A \cup A^*}} \{z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{in}(G^+_{A^*})\}\right\} < UB \end{cases} \tag{4.20}$$

As a consequence of (4.19) or (4.20), p_{1n} is a candidate for a robust shortest path in $G^+_{A^*}$. Then, *sol* and UB are updated to p_{1n} and $RCaux$, respectively, and the paths in list X that do not satisfy (4.17) for the new UB are discarded.

The pseudo-code of the method described above is given in Algorithm 11.

Computational time complexity order Let $\overline{m} = m + m^*$ be the number of arcs in $G^+_{A^*}$, with m^* the number of arcs added to G, and W^+_A be the maximum number of paths generated in $ST \cap P_{1i}(G^+_{A^*})$, $i \in V$. Like for Algorithms 8, 9 and 10, Algorithm 11 is divided in two parts.

The first part of Algorithm 11 is devoted to compute $\mathcal{T}^{s_u}_n(G^+_{A^*})$ and $LB^{s_u}_{in}(G^+_{A^*})$, $i \in V$, $u \in U_k$, in $O^a_1 = \mathcal{O}(k\overline{m})$ time for acyclic networks and in $O^c_1 = \mathcal{O}(k(\overline{m} + n \log n))$ time for general networks. These bounds are not affected by the remaining tasks. In fact, set $\widehat{U}_A^{A \cup A^*}$ is determined in $\mathcal{O}(k)$ time and the initial UB is set to $RC_{G^+_{A^*}}(\tilde{p})$ in $\mathcal{O}(kn)$ time. In the worst case, the calculation of set $\hat{A}$ demands the computation of $\mathcal{T}^{s_u}_1(G^+_{A^*})$ and $LB^{s_u}_{1i}(G^+_{A^*})$, $i \in V$, $u \in U_k$, with the same complexity for the homologous trees rooted at node n. Besides, checking the condition that defines $\hat{A}$, for each arc in A^*, demands $\mathcal{O}(1)$ operations, and, therefore, it is performed in $\mathcal{O}(m^*)$ time.

Algorithm 11: Finding the robust shortest path in $G^+_{A^*}$, given $\tilde{p}$, $RC_G(\tilde{p})$ and $LB^{s_u}_{1n}(G)$, $u \in U_k$

$\widehat{U}_A^{A\cup A^*} \leftarrow \emptyset$; $\hat{A} \leftarrow \emptyset$;
for $u \in U_k$ **do**
  Compute $\mathcal{T}_n^{s_u}(G^+_{A^*})$ and $LB^{s_u}_{in}(G^+_{A^*})$, $i \in V$;
  if $LB^{s_u}_{1n}(G^+_{A^*}) < LB^{s_u}_{1n}(G)$ **then** $\widehat{U}_A^{A\cup A^*} \leftarrow \widehat{U}_A^{A\cup A^*} \cup \{u\}$;
if $\widehat{U}_A^{A\cup A^*} = \emptyset$ **then** $RC_{G^+_{A^*}}(\tilde{p}) \leftarrow RC_G(\tilde{p})$;
else $RC_{G^+_{A^*}}(\tilde{p}) \leftarrow \max_{u\in U_k} RD^{s_u}_{G^+_{A^*}}(\tilde{p})$;
$UB \leftarrow RC_{G^+_{A^*}}(\tilde{p})$; $sol \leftarrow \tilde{p}$;
if $UB \neq 0$ **then**
  for $(i,j) \in A^*$ **do**
    for $u \in U_k$ **do**
      if $\mathcal{T}_1^{s_u}(G^+_{A^*})$ *was not computed yet* **then** Compute $\mathcal{T}_1^{s_u}(G^+_{A^*})$ and $LB^{s_u}_{1i}(G^+_{A^*})$, $i \in V$;
      if $RD^{s_u}_{G^+_{A^*}}(p^{1,s_u}_{1i}(G^+_{A^*}) \diamond \langle i,j\rangle \diamond p^{1,s_u}_{jn}(G^+_{A^*})) > UB$ **then**
        $\hat{A} \leftarrow \hat{A} \cup \{(i,j)\}$; **break**;
  if $\hat{A} \neq A^*$ ***or*** $\widehat{U}_A^{A\cup A^*} \neq \emptyset$ **then**
    $X \leftarrow \{\langle 1\rangle\}$;
    for $u \in U_k$ **do** $z^u_{G^+_{A^*}}(\langle 1\rangle) \leftarrow -LB^{s_u}_{1n}(G^+_{A^*})$;
    for $(i,j) \in A(\tilde{p})$ ***and*** $j \neq n$ **do**
      if $\max_{u\in U_k}\{z^u_{G^+_{A^*}}(\tilde{p}_{1i}) + c^{s_u}_{ij}(G^+_{A^*}) + LB^{s_u}_{jn}(G^+_{A^*})\} < UB$ **then**
        $X \leftarrow X \cup \{\tilde{p}_{1j}\}$;
        for $u \in U_k$ **do** $z^u_{G^+_{A^*}}(\tilde{p}_{1j}) \leftarrow z^u_{G^+_{A^*}}(\tilde{p}_{1i}) + c^{s_u}_{ij}(G^+_{A^*})$;
      else break;
    while $X \neq \emptyset$ **do**
      $p_{1i} \leftarrow$ first path in X; $X \leftarrow X - \{p_{1i}\}$; Compute $Ad_{G^+_{A^*}}(p_{1i}\,|\,\tilde{p})$;
      for $(i,j) \in Ad_{G^+_{A^*}}(p_{1i}\,|\,\tilde{p})$ **do**
        if $j = n$ **then**
          if $\widehat{U}_A^{A\cup A^*} \neq \emptyset$ **then**
            $aux1 \leftarrow \max_{u\in U_k\setminus \widehat{U}_A^{A\cup A^*}}\{z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{A^*})\}$;
            if $aux1 < UB$ **then**
            $RCaux \leftarrow \max\left\{aux1, \max_{u\in \widehat{U}_A^{A\cup A^*}}\{z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{A^*})\}\right\}$;
          else
            if $A(p_{1j}) \cap (A^*\setminus\hat{A}) \neq \emptyset$ **then** $RCaux \leftarrow \max_{u\in U_k}\{z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{A^*})\}$;
          if $RCaux < UB$ **then**
            $UB \leftarrow RCaux$; $sol \leftarrow p_{1i} \diamond \langle i,j\rangle$;
            for $p_{1i'} \in X$ **do**
              for $u \in U_k$ **do**
                if $z^u_{G^+_{A^*}}(p_{1i'}) + LB^{s_u}_{i'n}(G^+_{A^*}) \geq UB$ **then**
                  $X \leftarrow X - \{p_{1i'}\}$; **break**;
        else
          if $\max_{u\in U_k}\{z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{A^*}) + LB^{s_u}_{jn}(G^+_{A^*})\} < UB$ **then**
            $p_{1j} \leftarrow p_{1i} \diamond \langle i,j\rangle$; $X \leftarrow X \cup \{p_{1j}\}$;
            for $u \in U_k$ **do** $z^u_{G^+_{A^*}}(p_{1j}) \leftarrow z^u_{G^+_{A^*}}(p_{1i}) + c^{s_u}_{ij}(G^+_{A^*})$;
return *sol*;

The second part of Algorithm 11 concerns the construction of ST, which is analogous to Algorithm 10 in terms of the labeling and the two first extension rules, with complexities adapted for n nodes, $\overline{m}$ arcs and k scenarios. However, the third extension rule is different and may require more effort than in the previous algorithm. In fact, checking (4.19) requires the intersection $A(p_{1n}) \cap (A^* \backslash \hat{A})$, with $\mathcal{O}(n + m^*)$ operations [8] and setting $RCaux$ is done in $\mathcal{O}(k)$ time. This is also the complexity for checking (4.20). Hence, the third rule is performed in $\mathcal{O}(n + m^* + k)$ time. Because repeating the tests (4.17) may require $\mathcal{O}(knW_A^+)$ operations, the second stage is done in $O_2 = \mathcal{O}((m^* + knW_A^+)n^2W_A^+)$ time.

In conclusion, Algorithm 11 performs in $\mathcal{O}(\max\{m^*, knW_A^+\}n^2W_A^+)$, for any type of network, given that $\log n \ll n$ and $\overline{m} < n^2$.

Examples In the following, the set of arcs of the network G_5 in Figure 4.1 is extended to three new sets. For each of them, Algorithm 11 is applied, in order to return a robust shortest path in $(G_5)^+_{A^*}$. The first two examples cover the case $\widehat{U}_A^{A \cup A^*} = \emptyset$, while, in the third, $\widehat{U}_A^{A \cup A^*} \neq \emptyset$. In the first example, the arc $(2, 3)$ is inserted in G_5, and it belongs to $\hat{A}$, while in the second, besides $(2, 3)$, the arc $(3, 6)$, which is not in $\hat{A}$, is inserted in G_5 as well. The third example includes arc $(4, 7)$ in the previous network, in order to have $\widehat{U}_A^{A \cup A^*} \neq \emptyset$.

Case 1 Let $(G_5)^+_{(2,3)}$ be the network in Figure 4.16.

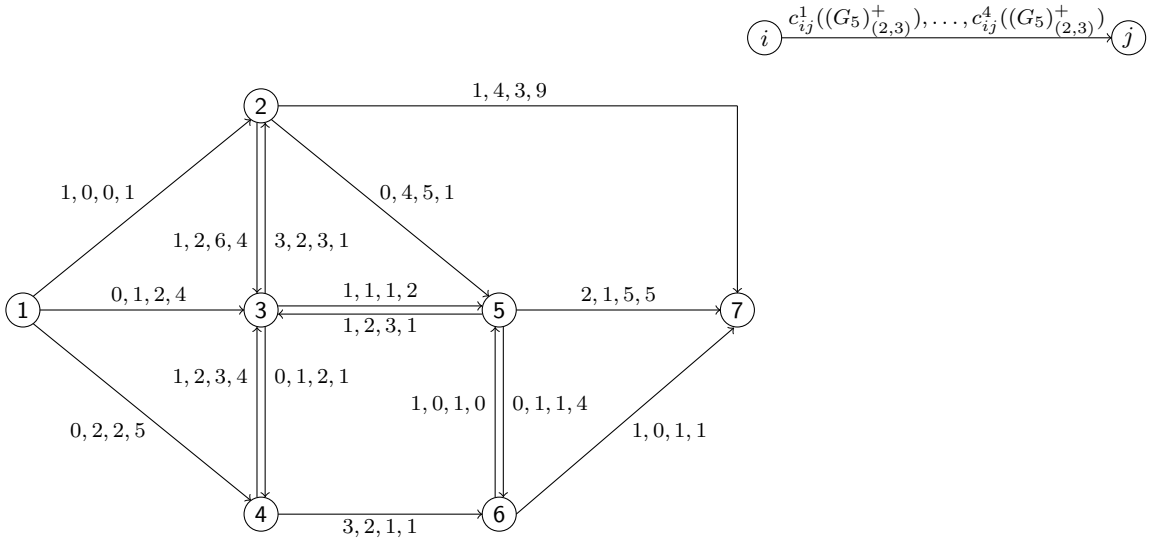


Figure 4.16: Network $(G_5)^+_{(2,3)}$

When arc $(2, 3)$ is inserted in network G_5, the shortest path trees rooted at node 7 do not

change. Hence, Figures 4.3 and 4.7.(a), (b) are taken into consideration. As,

$$LB^{u}_{17}((G_5)^{+}_{(2,3)}) = LB^{u}_{17}(G_5)\,,\ u \in U_4,$$

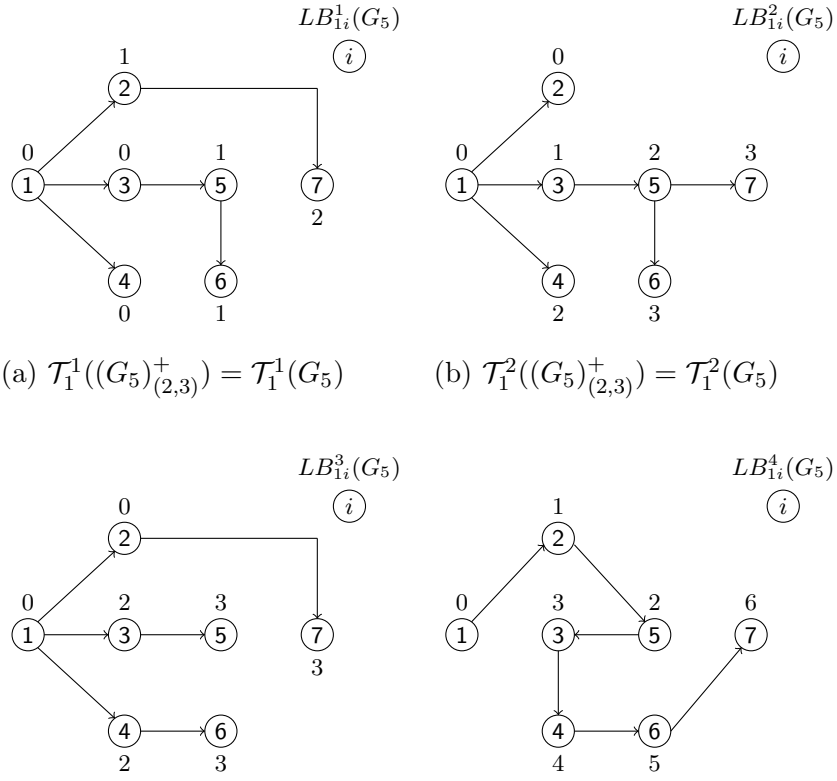
it follows that

$$\widehat{U}_A^{A\cup\{(2,3)\}} = \emptyset.$$

Then, UB and sol start with

$$UB = RC_{(G_5)^{+}_{(2,3)}}(\tilde{p}) = RC_{G_5}(\tilde{p}) = 2 \quad \text{and} \quad sol = \tilde{p} = \langle 1,4,6,7\rangle.$$

Since $UB \neq 0$, set $\hat{A}$ is determined. For this calculation, the shortest path trees rooted at node 1 in $(G_5)^{+}_{(2,3)}$ are needed, which coincide with the homologous trees in G_5. These trees are depicted in Figure 4.17.



(a) $\mathcal{T}^1_1((G_5)^{+}_{(2,3)}) = \mathcal{T}^1_1(G_5)$ (b) $\mathcal{T}^2_1((G_5)^{+}_{(2,3)}) = \mathcal{T}^2_1(G_5)$

(c) $\mathcal{T}^3_1((G_5)^{+}_{(2,3)}) = \mathcal{T}^3_1(G_5)$ (d) $\mathcal{T}^4_1((G_5)^{+}_{(2,3)}) = \mathcal{T}^4_1(G_5)$

Figure 4.17: Shortest path trees rooted at node 1 in G_5 and $(G_5)^{+}_{(2,3)}$

The only arc added to G_5, $(2,3)$, is now analyzed, satisfying

$$RD^1_{(G_5)^{+}_{(2,3)}}\left(p^{1,1}_{12}((G_5)^{+}_{(2,3)}) \diamond \langle 2,3\rangle \diamond p^{1,1}_{37}((G_5)^{+}_{(2,3)})\right) = RD^1_{(G_5)^{+}_{(2,3)}}(\langle 1,2,3,5,6,7\rangle) = 2 \leq UB,$$

$$RD^2_{(G_5)^+_{(2,3)}}(p^{1,2}_{12}((G_5)^+_{(2,3)}) \diamond \langle 2,3\rangle \diamond p^{1,2}_{37}((G_5)^+_{(2,3)})) = RD^2_{(G_5)^+_{(2,3)}}(\langle 1,2,3,5,7\rangle) = 1 \leq UB$$

and

$$RD^3_{(G_5)^+_{(2,3)}}(p^{1,3}_{12}((G_5)^+_{(2,3)}) \diamond \langle 2,3\rangle \diamond p^{1,3}_{37}((G_5)^+_{(2,3)})) = RD^3_{(G_5)^+_{(2,3)}}(\langle 1,2,3,5,6,7\rangle) = 6 > UB.$$

The analysis of $(2,3)$ halts, as $\hat{A} = \{(2,3)\}$. Hence, since $\widehat{U}_A^{A\cup\{(2,3)\}} = \emptyset$ and $\hat{A} = A^*$, Algorithm 11 returns $\tilde{p}$ as the robust shortest path of $(G_5)^+_{(2,3)}$.

Case 2 Consider now network $(G_5)^+_{A^*}$, with $A^* = \{(2,3),(3,6)\}$, in Figure 4.18.

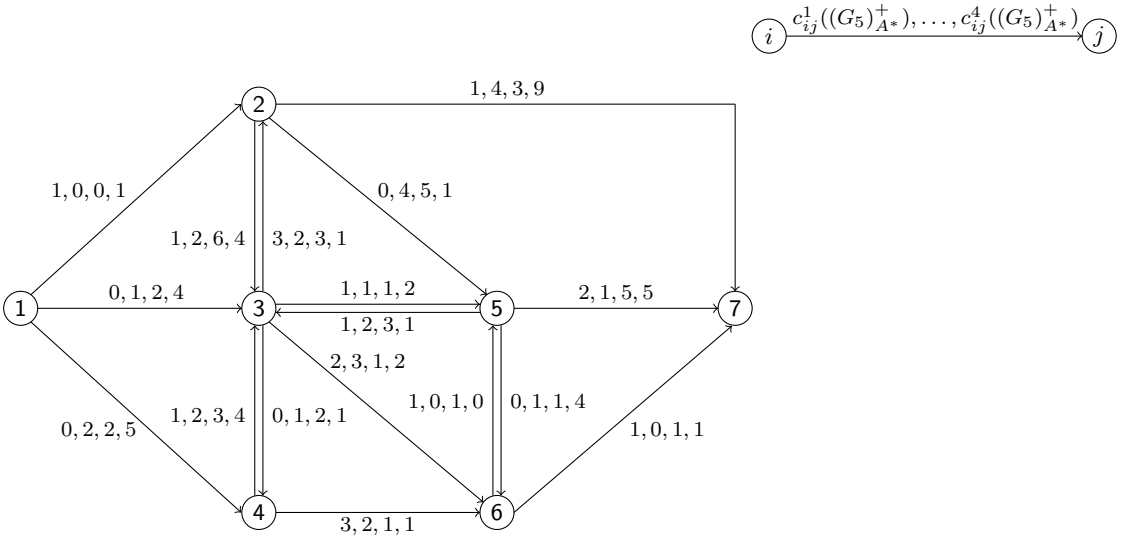


Figure 4.18: Network $(G_5)^+_{A^*}$, with $A^* = \{(2,3),(3,6)\}$

The tree $\mathcal{T}^3_7((G_5)^+_{A^*})$ is shown in Figure 4.19, because it is different from $\mathcal{T}^3_7(G_5)$.

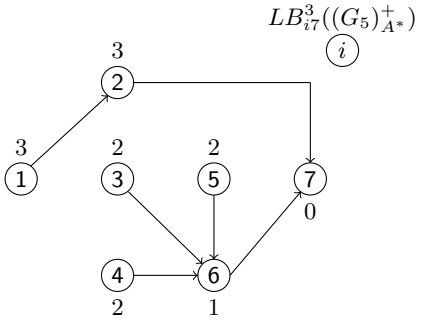


Figure 4.19: $\mathcal{T}^3_7((G_5)^+_{A^*})$, with $A^* = \{(2,3),(3,6)\}$

The remaining trees are still the same, as shown in Figures 4.3 and 4.7.(a). Because $p^{1,3}((G_5)^+_{A^*}) = p^{1,3}(G_5)$,

$$LB^u_{17}((G_5)^+_{A^*}) = LB^u_{17}(G_5)\,,\ u \in U_4,$$

and, therefore,

$$\widehat{U}^{A\cup A^*}_A = \emptyset.$$

Moreover, UB and sol are initialized with

$$UB = RC_{(G_5)^+_{A^*}}(\tilde{p}) = RC_{G_5}(\tilde{p}) = 2 \quad \text{and} \quad sol = \tilde{p} = \langle 1,4,6,7\rangle.$$

Since $UB \neq 0$, set $\hat{A}$ is determined. It can be noted that the shortest path trees rooted at node 1 in $(G_5)^+_{A^*}$ and G_5 are the same for all scenarios – Figure 4.17.

Analogously to Case 1, for arc $(2,3)$, the condition that defines $\hat{A}$ is not satisfied for scenarios 1 and 2. Because $\mathcal{T}^3_7((G_5)^+_{A^*}) \neq \mathcal{T}^3_7(G_5)$, the condition is checked for scenario 3, and

$$RD^3_{(G_5)^+_{A^*}}\left(p^{1,3}_{12}((G_5)^+_{A^*}) \diamond \langle 2,3\rangle \diamond p^{1,3}_{37}((G_5)^+_{A^*})\right) = RD^3_{(G_5)^+_{A^*}}(\langle 1,2,3,6,7\rangle) = 5 > UB.$$

Hence, it is concluded that $(2,3) \in \hat{A}$. For arc $(3,6)$,

$$RD^u_{(G_5)^+_{A^*}}\left(p^{1,u}_{13}((G_5)^+_{A^*}) \diamond \langle 3,6\rangle \diamond p^{1,u}_{67}((G_5)^+_{A^*})\right) = RD^u_{(G_5)^+_{A^*}}(\langle 1,3,6,7\rangle) = 1 \leq UB\ ,\ u \in U_3$$

and

$$RD^4_{(G_5)^+_{A^*}}\left(p^{1,4}_{13}((G_5)^+_{A^*}) \diamond \langle 3,6\rangle \diamond p^{1,4}_{67}((G_5)^+_{A^*})\right) = RD^4_{(G_5)^+_{A^*}}(\langle 1,2,5,3,6,7\rangle) = 0 \leq UB.$$

Therefore, $(3,6) \notin \hat{A}$, and, consequently,

$$\hat{A} = \{(2,3)\}.$$

As $\hat{A} \neq A^*$, Algorithm 11 starts to construct ST, by computing

$$X = \left\{\langle 1\rangle\right\} \cup \left\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in \{(1,4),(4,6)\} \text{ satisfy (4.18)}\right\}.$$

From the label

$$z_{(G_5)^+_{A^*}}(\langle 1\rangle) = (-2,-3,-3,-6),$$

one concludes that arc $(1,4)$ extends $\langle 1\rangle$, because

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1\rangle) + c^u_{14}((G_5)^+_{A^*}) + LB^u_{47}((G_5)^+_{A^*})\} = 1 < UB,$$

and the label

$$z_{(G_5)^+_{A^*}}(\langle 1,4\rangle) = (-2,-1,-1,-1)$$

is calculated. However, arc $(4,6)$ cannot be added to $\langle 1,4\rangle$, since

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,4\rangle) + c^u_{46}((G_5)^+_{A^*}) + LB^u_{67}((G_5)^+_{A^*})\} = 2 \geq UB.$$

Therefore, the initial X is

$$X = \{\langle 1\rangle, \langle 1,4\rangle\}.$$

Figure 4.20 shows the ST obtained for $(G_5)^+_{A^*}$ by Algorithm 11.

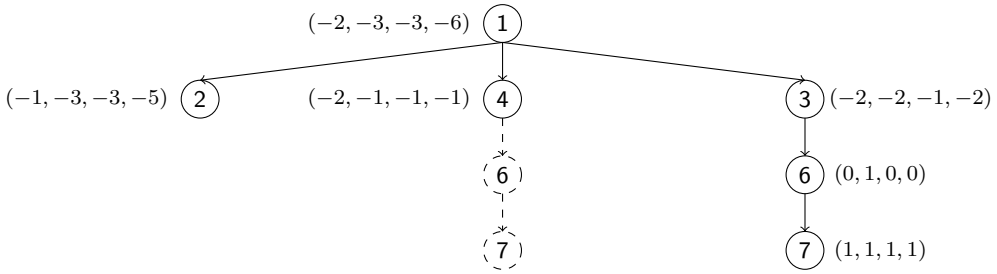


Figure 4.20: ST produced by Algorithm 11 for $(G_5)^+_{A^*}$, with $A^* = \{(2,3),(3,6)\}$

The possible arcs for extending $\langle 1\rangle$ belong to set

$$Ad_{(G_5)^+_{A^*}}(\langle 1\rangle \,|\, \tilde{p}) = \{(1,j) \in (A \cup A^*)\backslash(\hat{A} \cup A(\tilde{p}))\} = \{(1,2),(1,3)\}.$$

Both arcs can be added to $\langle 1\rangle$, since

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1\rangle) + c^u_{1i}((G_5)^+_{A^*}) + LB^u_{i7}((G_5)^+_{A^*})\} = 1 < UB, \;\; i = 2,3.$$

Then, the labels

$$z_{(G_5)^+_{A^*}}(\langle 1,2\rangle) = (-1,-3,-3,-5) \;\text{ and }\; z_{(G_5)^+_{A^*}}(\langle 1,3\rangle) = (-2,-2,-1,-2)$$

are created, and X is updated to

$$X = \{\langle 1,4\rangle, \langle 1,2\rangle, \langle 1,3\rangle\}.$$

Next, $\langle 1,4\rangle$ is selected, with

$$Ad_{(G_5)^+_{A^*}}(\langle 1,4\rangle \,|\, \tilde{p}) = \{(4,j) \in (A \cup A^*)\backslash(\hat{A} \cup A(\tilde{p})) : j \neq 1\} = \{(4,3)\}.$$

However,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,4\rangle) + c^u_{43}((G_5)^+_{A^*}) + LB^u_{37}((G_5)^+_{A^*})\} = 6 \geq UB,$$

which means $(4,3)$ cannot extend $\langle 1,4\rangle$. Afterwards, path $\langle 1,2\rangle$ is selected, with

$$Ad_{(G_5)^+_{A^*}}(\langle 1,2\rangle \,|\, \tilde{p}) = \{(2,j) \in (A \cup A^*)\backslash\hat{A} : j \neq 1\} = \{(2,5),(2,7)\},$$

but, no arc in this set can extend $\langle 1,2\rangle$. In fact, for arc $(2,5)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,2\rangle)+c^u_{25}((G_5)^+_{A^*})+LB^u_{57}((G_5)^+_{A^*})\}=4\geq UB.$$

For arc $(2,7)$, since $\widehat{U}_A^{A\cup A^*}=\emptyset$, (4.19) must be checked, but its first condition is not satisfied, since

$$A(\langle 1,2,7\rangle)\cap(A^*\backslash\hat{A})=A(\langle 1,2,7\rangle)\cap\{(3,6)\}=\emptyset.$$

Now, path $\langle 1,3\rangle$ remains to be extended, with

$$Ad_{(G_5)^+_{A^*}}(\langle 1,3\rangle\,|\,\tilde{p})=\{(3,j)\in(A\cup A^*)\backslash\hat{A}:j\neq 1\}=\{(3,2),(3,4),(3,5),(3,6)\}.$$

None of the arcs above can extend $\langle 1,3\rangle$, except $(3,6)$. In fact, for arcs $(3,2)$, $(3,4)$ and $(3,5)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,3\rangle)+c^u_{3i}((G_5)^+_{A^*})+LB^u_{i7}((G_5)^+_{A^*})\}=\begin{cases}5\,, & i=2\\ 3\,, & i=4\\ 4\,, & i=5\end{cases}$$

i.e.,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,3\rangle)+c^u_{3i}((G_5)^+_{A^*})+LB^u_{i7}((G_5)^+_{A^*})\}\geq UB,\;\; i=2,4,5.$$

Nevertheless, for arc $(3,6)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,3\rangle)+c^u_{36}((G_5)^+_{A^*})+LB^u_{67}((G_5)^+_{A^*})\}=1<UB,$$

Then, the label

$$z_{(G_5)^+_{A^*}}(\langle 1,3,6\rangle)=(0,1,0,0),$$

is calculated, and X is updated to

$$X=\{\langle 1,3,6\rangle\}.$$

The path above is the only one left for extension, with

$$Ad_{(G_5)^+_{A^*}}(\langle 1,3,6\rangle\,|\,\tilde{p})=\{(6,j)\in(A\cup A^*)\backslash\hat{A}:j\notin\{1,3\}\}=\{(6,5),(6,7)\}.$$

Arc $(6,5)$ is not added to $\langle 1,3,6\rangle$, as

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,3,6\rangle)+c^u_{65}((G_5)^+_{A^*})+LB^u_{57}((G_5)^+_{A^*})\}=4\geq UB,$$

however, arc $(6,7)$ extends $\langle 1,3,6\rangle$, because it satisfies (4.19),

$$\begin{cases}A(\langle 1,3,6,7\rangle)\cap(A^*\backslash\hat{A})=\{(3,6)\}\neq\emptyset\\ \\ RCaux=\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,3,6\rangle)+c^u_{67}((G_5)^+_{A^*})\}=1<UB.\end{cases}$$

Since there are no other paths in X to analyze, $\langle 1,3,6,7\rangle$ is returned as the robust shortest path in $(G_5)^+_{A^*}$, with $RC_{(G_5)^+_{A^*}}(\langle 1,3,6,7\rangle)=1$.

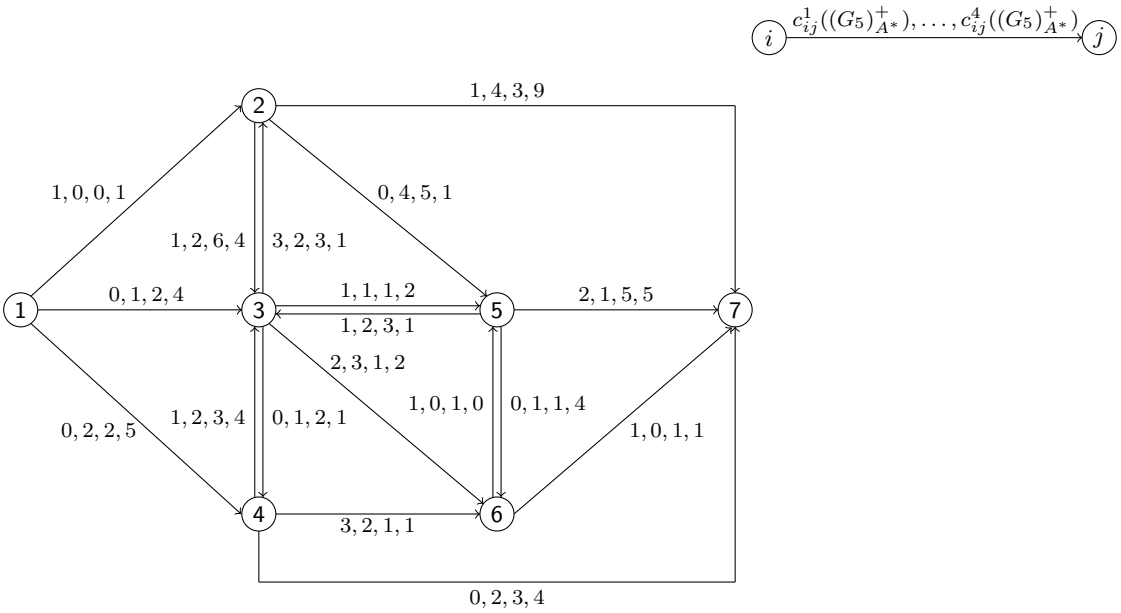


Figure 4.21: Network $(G_5)^+_{A^*}$, with $A^* = \{(2,3),(3,6),(4,7)\}$

Case 3 Consider now the network $(G_5)^+_{A^*}$, with $A^* = \{(2,3),(3,6),(4,7)\}$, in Figure 4.21. The shortest path trees in $(G_5)^+_{A^*}$ rooted at node 7 for scenarios 2, 3 and 4, in Figures 4.7.(a), 4.19 and 4.3.(b), do not change. The tree $\mathcal{T}_7^1(G^+_{A^*})$ is depicted in Figure 4.22.(a). Then,

$$\widehat{U}_A^{A\cup A^*} = \{u \in U_4 : LB_{17}^u((G_5)^+_{A^*}) < LB_{17}^u(G_5)\} = \{1\}.$$

Algorithm 11 sets the initial UB and sol,

$$UB = RC_{(G_5)^+_{A^*}}(\tilde{p}) = \max_{u\in U_4} RD^u_{(G_5)^+_{A^*}}(\tilde{p}) = 4 \quad \text{and} \quad sol = \tilde{p} = \langle 1,4,6,7\rangle.$$

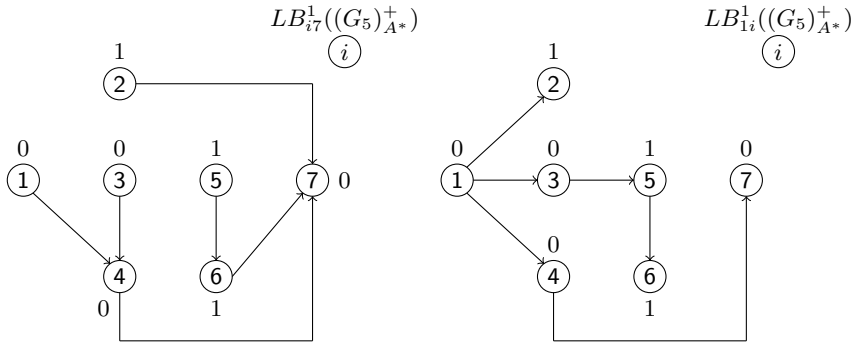
Since $UB \neq 0$, set $\hat{A}$ is determined. With this goal, the shortest path trees rooted at node 1 in $(G_5)^+_{A^*}$ must be considered. For scenarios 2, 3 and 4, these trees are depicted in Figures 4.17.(b), (c) and (d). The tree $\mathcal{T}_1^1(G^+_{A^*})$ is represented in Figure 4.22.(b).

In the following, the condition that defines $\hat{A}$ is tested for the arcs of A^*. Starting with arc $(2,3)$, the condition is not satisfied for scenario 1, because

$$RD^1_{(G_5)^+_{A^*}}\left(p_{12}^{1,1}((G_5)^+_{A^*}) \diamond \langle 2,3\rangle \diamond p_{37}^{1,1}((G_5)^+_{A^*})\right) = RD^1_{(G_5)^+_{A^*}}(\langle 1,2,3,4,7\rangle) = 2 \leq UB.$$

As in the previous cases, the condition does not hold for scenario 2, but it is satisfied for scenario 3, which means $(2,3) \in \hat{A}$. For arc $(3,6)$, the condition is not satisfied for scenario 1, since

$$RD^1_{(G_5)^+_{A^*}}\left(p_{13}^{1,1}((G_5)^+_{A^*}) \diamond \langle 3,6\rangle \diamond p_{67}^{1,1}((G_5)^+_{A^*})\right) = RD^1_{(G_5)^+_{A^*}}(\langle 1,3,6,7\rangle) = 3 \leq UB.$$



(a) $\mathcal{T}^1_7((G_5)^+_{A^*})$ (b) $\mathcal{T}^1_1((G_5)^+_{A^*})$

Figure 4.22: Shortest path trees rooted at nodes 7 and 1 in $(G_5)^+_{A^*}$ for scenario 1, with $A^* = \{(2,3),(3,6),(4,7)\}$

Like in Case 2, the condition is neither satisfied for the remaining scenarios. Hence $(3,6) \notin \hat{A}$. The new arc $(4,7)$ remains to be analyzed, for which,

$$RD^u_{(G_5)^+_{A^*}}(p^{1,u}_{14}\big((G_5)^+_{A^*}\big) \diamond \langle 4,7\rangle\big) = RD^u_{(G_5)^+_{A^*}}(\langle 1,4,7\rangle) = \begin{cases} 0\,, & u=1 \\ 1\,, & u=2 \\ 2\,, & u=3 \end{cases}$$

i.e.,

$$RD^u_{(G_5)^+_{A^*}}\big(p^{1,u}_{14}((G_5)^+_{A^*}) \diamond \langle 4,7\rangle\big) \leq UB, \;\; u \in U_3,$$

and

$$RD^4_{(G_5)^+_{A^*}}\big(p^{1,4}_{14}((G_5)^+_{A^*}) \diamond \langle 4,7\rangle\big) = RD^4_{(G_5)^+_{A^*}}(\langle 1,2,5,3,4,7\rangle) = 2 \leq UB.$$

Hence, $(4,7) \notin \hat{A}$, and one concludes that

$$\hat{A} = \{(2,3)\}.$$

Since $\widehat{U}^{A\cup A^*}_A \neq \emptyset$, ST has to be developed and starts with the paths of

$$X = \big\{\langle 1\rangle\big\} \cup \big\{\tilde{p}_{1i} \diamond \langle i,j\rangle : \tilde{p}_{1i} \in ST \text{ and } (i,j) \in \{(1,4),(4,6)\} \text{ satisfy (4.18)}\big\}.$$

From the label

$$z_{(G_5)^+_{A^*}}(\langle 1\rangle) = (0,-3,-3,-6),$$

one concludes that path $\langle 1,4\rangle$ belongs to ST, because

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1\rangle) + c^u_{14}((G_5)^+_{A^*}) + LB^u_{47}((G_5)^+_{A^*})\} = 1 < UB,$$

and the label

$$z_{G_A^+}(\langle 1,4\rangle) = (0,-1,-1,-1)$$

is calculated. However, path $\langle 1,4,6\rangle$ does not belong to ST, because

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,4\rangle) + c^u_{46}((G_5)^+_{A^*}) + LB^u_{67}((G_5)^+_{A^*})\} = 4 \geq UB.$$

Therefore, the initial X is

$$X = \{\langle 1\rangle, \langle 1,4\rangle\},$$

from which Algorithm 11 produces the ST depicted in Figure 4.23.

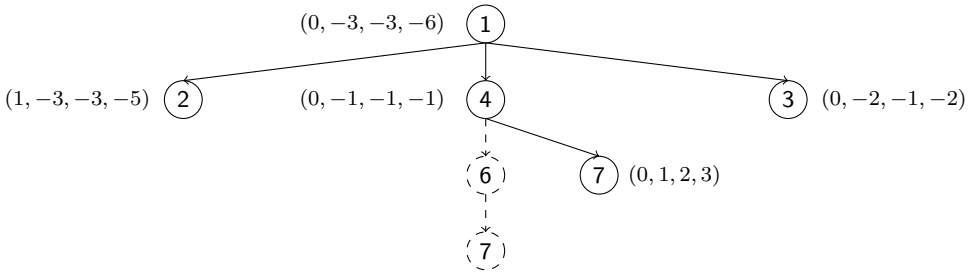


Figure 4.23: ST produced by Algorithm 11 for $(G_5)^+_{A^*}$, with $A^* = \{(2,3),(3,6),(4,7)\}$

The arcs of

$$Ad_{(G_5)^+_{A^*}}(\langle 1\rangle \,|\, \tilde{p}) = \{(1,j) \in (A \cup A^*)\backslash(\hat{A} \cup A(\tilde{p}))\} = \{(1,2),(1,3)\}$$

extend $\langle 1\rangle$, since, for arc $(1,2)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1\rangle) + c^u_{12}((G_5)^+_{A^*}) + LB^u_{27}((G_5)^+_{A^*})\} = 2 < UB$$

and, for arc $(1,3)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1\rangle) + c^u_{13}((G_5)^+_{A^*}) + LB^u_{37}((G_5)^+_{A^*})\} = 1 < UB.$$

Then, the labels

$$z_{(G_5)^+_{A^*}}(\langle 1,2\rangle) = (1,-3,-3,-5) \quad \text{and} \quad z_{(G_5)^+_{A^*}}(\langle 1,3\rangle) = (0,-2,-1,-2)$$

are calculated. Afterwards, X is updated to

$$X = \{\langle 1,4\rangle, \langle 1,2\rangle, \langle 1,3\rangle\},$$

and $\langle 1,4\rangle$ is the first path picked for extension, with

$$Ad_{(G_5)^+_{A^*}}(\langle 1,4\rangle \,|\, \tilde{p}) = \{(4,j) \in (A \cup A^*)\backslash(\hat{A} \cup A(\tilde{p})) : j \neq 1\} = \{(4,3),(4,7)\}.$$

Arc $(4,3)$ does not extend $\langle 1,4\rangle$, since

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,4\rangle)+c^u_{43}((G_5)^+_{A^*})+LB^u_{37}((G_5)^+_{A^*})\}=6\geq UB.$$

Nevertheless, recalling that $\widehat{U}_A^{A\cup A^*}=\{1\}$, arc $(4,7)$ can extend $\langle 1,4\rangle$, since (4.20) is satisfied,

$$\begin{cases} aux1=\max_{u\in\{2,3,4\}}\{z^u_{(G_5)^+_{A^*}}(\langle 1,4\rangle)+c^u_{47}((G_5)^+_{A^*})\}=3<UB \\ \\ RCaux=\max\{aux1,z^1_{(G_5)^+_{A^*}}(\langle 1,4\rangle)+c^1_{47}((G_5)^+_{A^*})\}=3<UB. \end{cases}$$

Then, UB and sol are updated to

$$UB=RCaux=3 \quad \text{and} \quad sol=\langle 1,4,7\rangle.$$

Path $\langle 1,2\rangle$ is the next to be scanned, with

$$Ad_{(G_5)^+_{A^*}}(\langle 1,2\rangle\,|\,\tilde{p})=\{(2,j)\in(A\cup A^*)\backslash\hat{A}:j\neq 1\}=\{(2,5),(2,7)\}.$$

None of these arcs extends $\langle 1,2\rangle$, because, for arc $(2,5)$,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,2\rangle)+c^u_{25}((G_5)^+_{A^*})+LB^u_{57}((G_5)^+_{A^*})\}=4\geq UB$$

and, for arc $(2,7)$, the first condition of (4.20) is not satisfied, since

$$aux1=\max_{u\in\{2,3,4\}}\{z^u_{(G_5)^+_{A^*}}(\langle 1,2\rangle)+c^u_{27}((G_5)^+_{A^*})\}=4\geq UB.$$

Afterwards, no extension is possible from $\langle 1,3\rangle$, with

$$Ad_{(G_5)^+_{A^*}}(\langle 1,3\rangle\,|\,\tilde{p})=\{(3,j)\in(A\cup A^*)\backslash\hat{A}:j\neq 1\}=\{(3,2),(3,4),(3,5),(3,6)\}.$$

In fact,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,3\rangle)+c^u_{3i}((G_5)^+_{A^*})+LB^u_{i7}((G_5)^+_{A^*})\}=\begin{cases} 5\,, & i=2 \\ 3\,, & i=4,6 \\ 4\,, & i=5 \end{cases}$$

i.e.,

$$\max_{u\in U_4}\{z^u_{(G_5)^+_{A^*}}(\langle 1,3\rangle)+c^u_{3i}((G_5)^+_{A^*})+LB^u_{i7}((G_5)^+_{A^*})\}\geq UB,\ \ i=2,4,5,6.$$

Since there are no other paths in X to scan, $\langle 1,4,7\rangle$ is returned as the robust shortest path in $(G_5)^+_{A^*}$, with $RC_{(G_5)^+_{A^*}}(\langle 1,4,7\rangle)=3$.

4.4 Conclusions

Chapter 4 presented four algorithms to reoptimize the robust shortest path in G after deleting or including some scenarios or some arcs. The introduced methods are able of computing a simple optimal solution in the transformed network, taking into account the original optimal solution, $\tilde{p}$. It is also assumed that $RC_G(\tilde{p})$ and the costs $LB_{1n}^{s_u}(G)$ are known.

For each type of perturbation introduced in G, conditions for which path $\tilde{p}$ maintained its optimality in the modified network were derived. For the deletion or inclusion of scenarios, $\tilde{p}$ was identified as the robust shortest path in $G_{S^*}^-$, when $RC_{G_{S^*}^-}(\tilde{p}) = 0$, and, in $G_{S^*}^+$, when $RC_{G_{S^*}^+}(\tilde{p}) = RC_G(\tilde{p})$. For the deletion or inclusion of arcs, $RC_{G_{A^*}^{\pm}}(\tilde{p}) = 0$ is one condition to $\tilde{p}$ preserve optimality. Otherwise, the identification of the scenarios indices for which the shortest $(1,n)$-path costs change from G to $G_{A^*}^{\pm}$, is initially required. If no index exists satisfying such property, path $\tilde{p}$ was identified as the robust shortest path in $G_{A^*}^-$, if $A(\tilde{p}) \cap A^* = \emptyset$, and, in $G_{A^*}^+$, if all of the arcs of A^* were robust 0-persistent.

When the conditions above are not satisfied for the transformed network in cause, it is possible to restrict the search for a new robust shortest path to a particular subset of simple $(1,n)$-paths. For that, a ST is constructed by means of extension rules applied to its paths, starting with the sub-paths of $\tilde{p}$ that exist in the transformed network. An upper-bound UB for the least robustness cost is also set in the latter network. In case of $G_{A^*}^-$, with $A(\tilde{p}) \cap A^* \neq \emptyset$, the initial UB is set to $\min\left\{RC_{G_{A^*}^-}(p^{1,s_u}(G_{A^*}^-)) : u \in U_k\right\}$. Otherwise, it is set to the new robustness cost of $\tilde{p}$. In general, the extension rules follow techniques used by the labeling and the hybrid algorithms in Chapter 2. One is the assignment of labels to each path in ST, as in the first, the other is the inclusion of arcs in ST that assure the simplicity of the obtained paths, as in the second for the deviation arcs. In case of $G_{A^*}^+$, the arcs avoid the identified robust 0-persistent arcs of A^* as well. The common technique, based on cost lower-bounds, is used to determine the first two extension rules. Specifically, they evaluate whether a given path p_{1i}, $i \in V\backslash\{n\}$, in ST can produce by extension a $(1,n)$-path in the transformed network with a robustness cost that can improve UB. The first rule considers path p_{1i} itself, and the second rule considers the addition of an arc (i,j), $j \neq n$, to p_{1i}. When $j = n$, the same kind of procedure applies, as a third extension rule, which, besides the latter property, is concerned with producing a $(1,n)$-path in the search set of the reoptimization method in cause.

The algorithms of the developed approaches were outlined and were shown to have time complexities depending on the modified network parameters, as well as on the maximum number of $(1,i)$-paths, $i \in V$, generated in ST. Moreover, in case of $G_{S^*}^-$ or $G_{A^*}^-$, they depend on the number of scenarios or arcs and nodes removed from G. The algorithms were exemplified for all cases.

Chapter 5

Concluding remarks

This final chapter is dedicated to summarize the main conclusions for each of the topics in this thesis. The steps of our research are described and are complemented with possible directions for future work.

In the first chapter, the main definitions and notation used along the thesis were introduced. The minmax regret robust shortest path problem was defined. Some of its properties were derived, and it was shown that it has a simple optimal solution, when G has no cycles with negative cost in any scenario. As a consequence, in the forthcoming chapters it was assumed that no such cycles exist in G nor in any of its modified versions.

Three algorithms for solving the robust shortest path problem were presented in Chapter 2 [39]. They are based on two main strategies, the labeling of paths rooted at node 1 and the ranking of $(1, n)$-paths in a particular scenario. The literature review showed that Murthy and Her [35], with a labeling method, and Yu and Yang [50], with a dynamic programming method, proposed to solve the problem exactly for discrete models. The first only considered the minmax shortest path problem, presenting an effective method. It was based on labeling paths rooted at node 1, for which dominance tests were applied and the extension to node n was analyzed by means of cost lower-bounds. The second considered both the absolute and the relative versions of robustness. However, their method was computationally heavy in general, especially for networks with a large number of scenarios or large cost upper-bounds.

The previous facts motivated the adaptation of Murthy and Her's method to the relative version of robustness, considering regret costs as the objective functions and skipping the dominance tests for $(1, n)$-paths, in order to spare computational effort. The new labeling algorithm, LA, clearly outperformed Yu and Yang's algorithm, YA, in the first set of computational experiments, even for small networks with a small number of scenarios. The next step was to design a strategy that could compete with LA. An alternative approach was ranking $(1, n)$-paths. For instance, with interval data, that method had been applied for the multicriteria

shortest path problem by Dias and Clímaco [16], and also for the robust shortest path problem by Montemanni and Gambardella [31], through handling the lower and the upper-limits of the cost intervals. For the discrete model, the ranking was applied, considering all possible cost scenarios. The first algorithmic version, RA, was presented, consisting in ranking simple $(1, n)$-paths by non-decreasing order of cost under a suitable scenario s_r. The cost upper-bound is reduced along the process, taking into account the best robustness cost of the ranked paths. Moreover, the ranking halts, whenever the least maximum robust deviation occurs in scenario s_r. Empirical tests revealed that this task was unstable as well as computationally hard in several cases.

One way to improve RA was the application of the pruning rule used in LA when ranking $(1, n)$-paths. This combination led to a second ranking approach, denominated hybrid algorithm, HA. The paths ranking was based on the deviation method introduced in [30], considering in each iteration all the deviation arcs from a $(1, i)$-path, $i \in V$, that can lead to optimal simple paths. This new technique allowed to skip a significant number of paths in the ranking.

The new algorithms have time complexities depending on the maximum number of paths generated in $P_{1i}(G)$, $i \in V$, for LA, and on the number of ranked paths for RA and HA. Empirical tests have shown that the new algorithms outperform YA, with HA and LA having similar performances. Both algorithms were able to solve the problem for a relatively large number of scenarios in reasonable time. Because of this fact, approximating continuous cost models, by means of discretizing the cost functions in a significant number of scenarios can be included in a possible future research. The study can also cover particular continuous cost functions, for which an exact solution can be found, such as piecewise linear functions.

In Chapter 3, preprocessing tools for reducing the size of G before finding a robust shortest path were developed and tested. These techniques aimed at identifying arcs or nodes robust 1-persistent, which belong to all optimal solutions, and robust 0-persistent, which do not belong to any of them. The study was restricted to robust 1-persistent arcs and robust 0-persistent nodes, as they lead to the best reduction of G. Only the case of interval data models has been approached, first, by Karasan, Pinar and Yaman [26], and, more recently, by Catanzaro, Labbé and Salazar-Neumann [13]. The latter work presented more efficient and more general results than the previous.

The preprocessing rules for the discrete case were derived, setting inequalities involving specific path robust deviations and cost lower-bounds. The first approach, denominated static and introduced in [40], used fixed lower-bounds, which depended on the paths $p^{1,s_u}(G)$, $u \in U_k$, with the least robustness cost in G. The arcs candidate to be robust 1-persistent belong to those paths, while the nodes candidate to be robust 0-persistent do not. Both algorithms are

polynomial in terms of time. In the empirical experiments, only very few robust 1-persistent arcs have been identified. On the contrary, the static identification of robust 0-persistent nodes was very effective, as the problem size was significantly reduced. In addition, combining static preprocessing with LA outperformed its application without preprocessing only for the networks with the highest density. This did not happen for HA, because the method was able to find optimal solutions by itself within few iterations for all the considered instances.

A new technique was explored, in order to improve the previous results. This second approach, denominated dynamic and introduced in [41], aimed to detect more arcs and nodes than with the static version. Nevertheless, the arcs candidate to be robust 1-persistent were still restricted to the paths $p^{1,s_u}(G)$, $u \in U_k$. The same inequalities to test arcs or nodes in the static algorithm were used. However, the cost lower-bounds and the sets of scanned elements were updated along the process, according with the least robustness cost of the computed $(1,n)$-paths. The dynamic search for robust 1-persistent arcs was as ineffective as the static. On the contrary, the improvement of the static procedure for robust 0-persistent nodes was accomplished with the dynamic algorithm, by allowing to improve the reduction of the instances in the empirical experiments. In order to further spare computational effort, the number of scenarios used for testing nodes was limited. The results showed that LA or HA after dynamic preprocessing run faster than after the static search. Combining dynamic preprocessing with LA, for the majority of the instances, or with HA, for networks with a large number of nodes, outperformed finding a robust shortest path in the original G.

The influence of robust 1-persistent arcs in detecting robust 0-persistent nodes was also studied. In fact, when some of those arcs is identified, the calculation of the path robust deviations in the test conditions for preprocessing nodes can be skipped. However, because detecting robust 1-persistent arcs was rare in most of the networks, the new approach was not efficient. In order to improve the reduction of G with the dynamic preprocessing, more conditions to find robust 1-persistent arcs deserve further investigation. One possible direction is to give priority to the dynamic detection of robust 0-persistent nodes and with the minimum lower-bound attained in that preprocessing, test candidates besides the arcs of the paths $p^{1,s_u}(G)$, $u \in U_k$. Another strategy is to consider an arc (i,j) robust 1-persistent in G, by identifying the remaining arcs with tail node i and with head node j as robust 0-persistent.

The fourth chapter focused on reoptimizing the robust shortest path, assuming the deletion or insertion of scenarios or of arcs in G. This topic was raised by Chapter 3, where the reduction of G had been treated, in the context of preserving the original robust shortest path of G, $\tilde{p}$. The purpose of Chapter 4 was to reoptimize the solution in the modified version of G, assuming that $\tilde{p}$, $RC_G(\tilde{p})$ and the costs $LB_{1n}^{s_u}(G)$, $u \in U_k$, are known. First, the conditions that ensure that $\tilde{p}$ is still optimal in G's modified version, were presented. When none of the previous cases

is guaranteed, a simple robust shortest path in the transformed network is searched in a specific subset of $(1, n)$-paths, with the least robustness cost in the modified network.

In the literature, no tools for reoptimizing the robust shortest path problem were found, so the idea explored for the search method was to combine techniques of LA and HA, both reported as the most efficient in Chapter 2. A strategy adopted to perform the search was to start to add one arc at a time from node 1, so that the obtained paths can produce potentially optimal simple paths in the transformed network. As a consequence, a search-tree, ST, was constructed, starting with the sub-paths of $\tilde{p}$, except itself, in the modified network. The labeling of the paths in ST was similar to LA, because repeating the sum of previous arc costs could be avoided, and when adding an arc with head node n, the robustness cost in the transformed network could be immediately obtained. The dominance tests of LA were not considered here. Therefore, the arcs to be added to ST were chosen similarly to the deviation arcs in HA.

The developed reoptimization algorithms have time complexities, depending on the maximum number of $(1, i)$-paths, $i \in V$, in ST. The complexity orders depended also on the number of removed scenarios or arcs when G was reduced. The algorithms were exemplified for a particular instance, considering all possible situations. Future research should include empirical tests, comparing the performances of the reoptimization algorithms with the direct application of LA or HA over the transformed versions of G. This study is important to decide what is the best approach to reoptimize the problem.

Bibliography

[1] R.K., Ahuja, T.L., Magnanti and J.B. Orlin. *Network Flows : Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] C. Archetti, L. Bertazzi and M. Speranza. Reoptimizing the 0-1 knapsack problem. *Discrete applied mathematics*, vol. 158(17), 2010.

[3] G. Ausiello, V. Bonifaci and B. Escoffier. Complexity and approximation in reoptimization. In B. Cooper and A. Sorbi Eds., editors, *Computability in Context: Computation and Logic in the Real World*, pages 101–129. Imperial College Press/World Scientific, 2011.

[4] G. Ausiello, B. Escoffier, J. Monnot and V. Paschos. Reoptimization of minimum and maximum traveling salesmans tours. *Journal of Discrete Algorithms*, 7(4):453–463, 2009.

[5] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 1:425–447, 1958.

[6] D. Bertsimas, D.B. Brown and C. Caramanis. Theory and Applications of Robust Optimization. *SIAM Review*, 53:464–501, 2011.

[7] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Math.Program., Ser.B*, 98:49–71, 2003.

[8] P. Bille, A. Pagh and R. Pagh. Fast evaluation of union-intersection expressions. Technical report, IT University of Copenhagen, Denmark, 2007.

[9] H. Böckenhauer, L. Forlizzi, J. Hromkovič, J. Kneis, J. Kupke, G. Proietti and P. Widmayer. On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research*, 2(2), 2007.

[10] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43(2):216–224, 1989.

[11] 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.

[12] A. Candia-Véjar, E. Álvarez Miranda, and N. Maculan. Minmax regret combinatorial optimization problems: An algorithmic perspective. *RAIRO Operations Research*, 45:101–129, 2011.

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

[14] J. Clímaco and E. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.

[15] C. Demetrescu, D. Eppstein, Z. Galil, and G. Italiano. Dynamic graph algorithms. In M. J. Atallah and M. Blanton, editors, *Algorithms and theory of computation handbook: general concepts and techniques*, pages 1–25 of Chapter 9. Chapman & Hall/CRC, 2010.

[16] 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.

[17] M. Desrochers and F. Soumis. A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, 35(2):242–254, 1988.

[18] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28:652–673, 1998.

[19] L. Ford. Network flow theory. Technical Report P-923, The Rand Corp., Santa Monica, CA, August 1956.

[20] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985.

[21] 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.

[22] V. Gabrel and C. Murat. Robust shortest path problems. Technical Report 7, Annales du LAMSADE, Paris, May 2007.

[23] V. Gabrel, C. Murat and A. Thiele. Recent advances in robust optimization and robustness: An overview, July 2012. Working paper.

[24] G. Gallo. Reoptimization procedures in shortest path problem. *Rivista di matematica per le scienze economiche e sociali*, 3(1):3–13, 1980.

[25] F. Guerriero and R. Musmanno. Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Application*, 11:589–613, 2001.

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

[27] P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Kluwer Academic Publishers, Boston, 1997.

[28] E. Q. V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984.

[29] 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.

[30] 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.

[31] R. Montemanni and L. Gambardella. An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31:1667–1680, 2004.

[32] 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.

[33] 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.

[34] E. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching, Part II*, volume 30, pages 285–292. Harvard University Press, 1959.

[35] I. Murthy and S.-S. Her. Solving min-max shortest path problems on a network. *Naval Research Logistics*, 39:669–683, 1992.

[36] S. Nguyen, S. Pallottino and M. Scutellà. A New Dual Algorithm for Shortest Path Reoptimization. In M. Gendreau and P. Marcotte, editors, *Series of Applied Optimization – Transportation and Network Analysis: Current Trends*, volume 63, pages 221–235, 2002. Springer US.

[37] S. Pallottino and M. Scutellà. A new algorithm for reoptimizing shortest paths when the arc costs change. *Operations Research Letters*, 31(2):149–160, 2003.

[38] 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.

[39] 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.

[40] 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 – Mathematics of Planet Earth: Energy and Climate Change (Dynamics, Games and Science)*, volume 2. Springer-Verlag, to appear, 2014.

[41] M. Pascoal and M. Resende. Dynamic preprocessing for the minmax regret robust shortest path problem with finite multi-scenarios. Technical Report 10, INESC-Coimbra, Coimbra, September 2014. (under revision).

[42] 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.

[43] A. Pessoa, L. Pugliese, F. Guerriero and M. Poss. Robust constrained shortest path problems under budgeted uncertainty. *Networks*, doi: 10.1002/net.21615, April, 2015.

[44] H. Rohnert. A dynamization of the all-pairs least cost problem. In K. Mehlhorn, editor, *Proceedings 2nd Symposium on Theoretical Aspects of Computer Science*, pages 279–286, 1985.

[45] B. Roy, Méthodologie multicritère d'aide à la décision. *Economica*, Paris, 1985.

[46] B. Roy, Robustness in operational research and decision aiding: A multi-faceted issue. *European Journal of Operational Research*, 200:629–638, 2010.

[47] H. Shachnai, G. Tamir and T. Tamir. A theory and algorithms for combinatorial reoptimization. In D. Fernández-Baca, editor, *LATIN 2012: Theoretical Informatics*, pages 618–630. Springer Science & Business Media, April 2012.

[48] A. J. V. Skriver and K. A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers & Operations Research*, 27(6):507–524, 2000.

[49] J. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.

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

List of notation

U_k : $\{1, \ldots, k\}$, $k > 1$.

$G(V, A, S)$: directed graph G with a set of nodes $V = \{1, \ldots, n\}$, a set of arcs A s.t. $A \subseteq \{(i, j) : i, j \in V \text{ and } i \neq j\}$, and a set of scenarios $S = \{s_u : u \in U_k\}$.

$V(p)\,(A(p))$: set of nodes (arcs) of path p.

$p \diamond q$: concatenation of paths p and q.

$P_{ij}(G)$: set of (i, j)-paths in G, $i, j \in V$.

$c_{ij}^{s_u}(G)$: cost of arc (i, j) under scenario s_u, $u \in U_k$, in G.

$c_G^{s_u}(p)$: cost of a path p under scenario s_u, $u \in U_k$, in G.

$p_{ij}^{l,s_u}(G)$: l-th shortest (i, j)-path of G, $i, j \in V$, in scenario s_u, $u \in U_k$.

$p^{l,s_u}(G)$: l-th shortest $(1, n)$-path of G in scenario s_u, $u \in U_k$.

$LB_{ij}^{s_u}(G)$: cost of the shortest (i, j)-path of G, $i, j \in V$, in scenario s_u, $u \in U_k$.

$\mathcal{T}_1^{s_u}(G)\,(\mathcal{T}_n^{s_u}(G))$: tree of the shortest $(1, i)$-paths ((i, n)-paths), $i \in V$, of G.

$RD_G^{s_u}(p)$: robust deviation of a $(1, n)$-path p under scenario s_u, $u \in U_k$, in G.

$RC_G(p)$: robustness cost of a $(1, n)$-path p in G.

$U_G(p)$: set of scenarios indices under which $RC_G(p)$ occurs.

$z_G(p_{1i})$: label associated with a $(1, i)$-path p_{1i}, $i \in V$.

$Z_G(P_{1i})$: set of labels for all the paths of $P_{1i}(G)$, $i \in V$.

q_{ij}^{p,s_r} : $(1,n)$-path formed by the $(1,i)$-sub-path of p followed by arc $(i,j) \in A$, and, then, by the shortest (j,n)-path under scenario s_r in G, $r \in U_k$.

$\bar{c}_{ij}^{s_r}(G)$: reduced cost of arc (i,j) in scenario s_r, $r \in U_k$.

$\mathcal{A}_G^{s_r}(i)$: set of arcs of G with tail node i sorted by non-decreasing order of the reduced costs with respect to scenario s_r, $r \in U_k$.

$\hat{\mathcal{A}}_G^{s_r}(p_{1i}, j)$: subset of arcs of $\mathcal{A}_G^{s_r}(i) \backslash \{(i,j)\}$, which extend the $(1,i)$-path p_{1i}, $i \in V$, to a simple $(1,n)$-path.

$U(i,j)$: set of scenarios for which the shortest $(1,n)$-paths of G contain arc (i,j).

S^* (A^*) : non-empty set of scenarios (arcs) removed from or added to G.

$G_{S^*}^-(G_{S^*}^+)$: subgraph (extension) of G with set of scenarios $S \backslash S^*$ $(S \cup S^*)$.

$G_{A^*}^-(G_{A^*}^+)$: subgraph (extension) of G with set of arcs $A \backslash A^*$ $(A \cup A^*)$.

$\widehat{P}_{1n}(G_{S^*}^-)$: set of $(1,n)$-paths of $G_{S^*}^-$, for which the robust deviation in G under some of its removed scenarios is not smaller than the optimal value of G.

$\widehat{U}^*$: set of the indices of the scenarios added to G, under which the robust deviation of the optimal solution of G exceeds its robustness cost.

$\widehat{P}_{1n}(G_{S^*}^+)$: set of $(1,n)$-paths of $G_{S^*}^+$, for which the robustness cost in G is smaller than their robustness cost in $G_{S^*}^+$.

$\widehat{U}_{A \backslash A^*}^{A}$ $(\widehat{U}_{A}^{A \cup A^*})$: set of the scenarios indices of G, for which the shortest $(1,n)$-paths cost change from G to $G_{A^*}^-$ $(G_{A^*}^+)$.

$\widehat{P}_{1n}(G_{A^*}^-)$: set of $(1,n)$-paths of $G_{A^*}^-$, for which the robust deviation in G under some scenario s_u, $u \in \widehat{U}_{A \backslash A^*}^{A}$, is not smaller than the optimal value of G.

$\hat{A}$: set of arcs added to G, which are robust 0-persistent in $G_{A^*}^+$.

$\bar{P}_{1n}(G_{A^*}^+)$: set of $(1,n)$-paths of $G_{A^*}^+$, which do not contain any arc in $\hat{A}$.

$\widehat{P}_{1n}(G_{A^*}^+)$: subset of paths in $\bar{P}_{1n}(G_{A^*}^+)$, for which the robust deviation under some scenario s_u, $u \in U_k \backslash \widehat{U}_{A}^{A \cup A^*}$, is smaller than their robustness cost in $G_{A^*}^+$.

$Ad_{G'}(p_{1i} \,|\, \tilde{p})$: set of arcs added to a $(1,i)$-path p_{1i}, $i \in V$, in the search tree that reoptimizes the optimal solution $\tilde{p}$ of G, with $G' = G_{S^*}^{\pm}$ or $G' = G_{A^*}^{\pm}$.