Bounds and Preprocessing for the Robust Resource Constrained Shortest Path Problem

Luigi Di Puglia Pugliese, Francesca Guerriero, Michael Poss

Bounds and Preprocessing for the Robust Resource Constrained Shortest Path Problem

Luigi Di Puglia Pugliese* and Francesca Guerriero†

Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende, Italy

Michael Poss‡

UMR CNRS 5506 LIRMM, Université de Montpellier 2, 161 rue Ada, 34392 Montpellier Cedex 5, France

Abstract

We address a robust version of the Resource Constrained Shortest Path Problem ($RCSPP$). We assume that the resource consumptions are uncertain and may take any value in the budgeted uncertainty set defined by Bertismas and Sim (2003,2004). We solve the problem to optimality through a dynamic programming approach well-tailored for the robust $RCSPP$, which adopts state-of-the-art resolution techniques for the deterministic counterpart. In particular, we formulate the robust Lagrangian relaxation and solve the corresponding dual problem. Upper and lower bounds information are used to fathom unpromising states. In addition, nodes and arcs removing procedures are defined by using robust resource consumption associated with forward and backward paths. The proposed dynamic programming is tested on instances inspired by the scientific literature for the $RCSPP$.

Keywords: Constrained shortest path, Robust optimization, Budgeted uncertainty, Dynamic programming.

1 Introduction

The Resource Constrained Shortest Path Problem ($RCSPP$) is defined over a directed graph $G(N, A)$ where N is the set of n nodes, and A is the set

*luigi.dipugliapugliese@unical.it, *Corresponding author*
†francesca.guerriero@unical.it
‡michael.poss@lirmm.fr

of m arcs. The set N contains two special nodes o and d that represent the source node and the destination node, respectively. A cost κ_a has to be paid for traversing the arc $a = (i, j) \in A$, whereas, a quantity of resource r_a has to be used along the arc $a = (i, j) \in A$. In the following, terms κ_a and κ_{ij} (r_a and r_{ij}) are used interchangeably. A path $p_j = (i_1 = o, \ldots, i_l = j)$ is an ordered sequence of l nodes from the source o to j. It can be viewed as an order sequence of $l-1$ arcs $((o, i_2), (i_2, i_3), \ldots, (i_{l-1}, i_l))$. The cost of a path p is defined as $\kappa(p) = \sum_{(i,j)\in p} \kappa_{ij}$, whereas the resource consumption along p is denoted as $w(p) = \sum_{(i,j)\in p} r_{ij}$. The aim of $RCSPP$ is to find a path p with the smallest cost, such that $w(p) \leq W$, where W is the maximum amount of available resource.

The scientific literature has paid great attention to $RCSPP$. Several papers have proposed efficient optimal solution approaches. Branch-and-bound, path ranking, and dynamic programming approaches are the most efficient frameworks to deal with the optimal solution of the $RCSPP$. We cite, among others, [3, 7] for branch-and-bound schemes, [9, 13] for dynamic programming approaches, and [19, 20] for path ranking methods. For all classes of solution strategies, the authors of these papers have used lower and upper bounds information, derived from the resolution of the Lagrangean dual relaxation to speed up the basic algorithms. In addition, network reduction techniques have been developed to eliminate nodes and arcs ensuring the optimality of the solution. For a complete survey of optimal solution strategies, lower and upper bounds computation and network reduction procedures, the reader is referred to [8].

We suppose herein that the resource consumptions are uncertain parameters that can take any values in the budgeted uncertainty set proposed in [4, 5]. Namely, given an integer Γ and two integer vectors $\overline{r}$ and $\hat{r}$, respectively describing the mean values of the resource consumption and the deviations, the budgeted uncertainty set can be defined as

$$\mathcal{U}^{\Gamma} = \left\{ r_a = \overline{r}_a + \delta_a \hat{r}_a : 0 \leq \delta_a \leq 1, \sum_{a \in A} \delta_a \leq \Gamma \right\}.$$

Given a fixed $r \in \mathcal{U}^{\Gamma}$, we define the resource consumption along a path p as $w(p, r)$. It is worth observing that, since $w(p, r)$ is a convex function, we can restrict ourselves without loss of generality to the set of extreme points of $\mathcal{U}^{\Gamma}$, thus assuming $\delta_a \in \{0, 1\}$ for each $a \in A$. In our robust context, the constraint related to the maximum resource consumption becomes

$$\max_{r \in \mathcal{U}^{\Gamma}} w(p, r) \leq W.$$

We denote the robust $RCSPP$ under budgeted uncertainty set as $\mathcal{U}^{\Gamma}$-$RCSPP$. The budgeted uncertainty set has been very widely in the robust combinatorial optimization literature for two main reasons. First, the set benefits from profound links with probabilistic constraints, which can also ease the choice of Γ, see initial bound from [5] and the extension to variable uncertainty in [17, 18]. Second, [4] prove in their seminal paper that the set keeps the tractability and

approximation ratios of the min max robust counterparts of many combinatorial optimization problems. These positive results have been extended to problems with robust constraints (as $\mathcal{U}^\Gamma$-$RCSPP$) independently by [2] and [10]. Ad-hoc algorithms have also been provided for specific robust combinatorial optimization problems with budgeted uncertainty, many of them relying on dynamic programming algorithms [1, 11, 15, 21]. Other approaches than dynamic programming have also been considered, see for instance the MILP reformulation proposed in [6] for the recoverable robust knapsack problem.

Problem $\mathcal{U}^\Gamma$-$RCSPP$ has been recently investigated in three papers. In [16], the authors address $\mathcal{U}^\Gamma$-$RCSPP$ as well as the version with time windows constraints ($\mathcal{U}^\Gamma$-$TWSPP$). Applying the approach from [10], they prove that the former problem is weakly $\mathcal{NP}$-hard. In contrast, they show that the latter problem is strongly $\mathcal{NP}$-hard. They also develop a general label-setting algorithm for both problems, based on the definition of new robust labels. The algorithm has an exponential running-time when Γ is part of the input. Numerical results are carried out for the $\mathcal{U}^\Gamma$-$TWSPP$. In [14], the authors study $\mathcal{U}^\Gamma$-$RCSPP$ as well as the version where the uncertainty set is an ellipsoid. They propose a pseudo-polynomial time algorithm for the second problem based on solving a sequence of deterministic $RCSPP$, similarly to the seminal result from [4]. They also present numerical results for both problems (with budgeted and ellipsoidal uncertainty). Finally, the authors of [12] apply a Lagrangian relaxation to a robust crew pairing problem ending up with a subproblem identical to $\mathcal{U}^\Gamma$-$RCSPP$.

The purpose of this paper is to build on the dynamic programming algorithm proposed in [16]. In that paper, the focus was $\mathcal{U}^\Gamma$-$TWSPP$ rather than $\mathcal{U}^\Gamma$-$RCSPP$, so that bounding and preprocessing techniques were not used. We fill the gap herein by showing how the classical graph reduction algorithms and bounds obtained from the Lagrangian relaxation of the problem (see [9]) can be extended to the robust context. We combine these techniques with the label-setting algorithm from [16] to obtain an algorithm tailored for $\mathcal{U}^\Gamma$-$RCSPP$. We conduct numerical experiments to assess the efficiency of the graph reduction as well as the reduction in solution times. We compare our approach with the classical iterative algorithm used previously by [14] and [12].

The paper is organized as follows. In section 2, we recall label-setting algorithm proposed in the deterministic context. In section 3, we review the two extensions to the robust context proposed by [16]. In section 4, we define robust Lagrangean relaxation along with hints on how to solve the dual counterpart. In addition, network reduction procedures based on robust information are defined. We show how to incorporate lower and upper bounds information in the dynamic programming solution scheme to speed up the search process. We describe solution strategies dealing with the robust shortest path problem used to determine robust bounds. Section 5 reports computational results on instances inspired by benchmark for the $RCSPP$. Conclusions are given in section 6. The appendix reports detailed results on the effect of preprocessing.

2 Deterministic label-setting algorithm

For completeness, we present below the well-known label-setting algorithm used in [9,13] as well as the preprocessing techniques used in [9]. A label $(\kappa(p_j), w(p_j))$ is associated with each path p_j. Labels corresponding to each node $\in V$ are stored in the set of labels S_i and $S = \cup_{i \in N} S_i$. Starting from the initial label $(0, 0)$ associated with path $p_o = (o)$, new labels are generated for each node $j : (o, j) \in A$. The generated labels are stored in the corresponding set S_j. A label $(\kappa(p_i), w(p_i))$ is selected and marked as permanent at each iteration, that is, the label is removed from S_i. To limit the number of generated labels, only non-dominated and feasible paths are considered, which is formalized in the following lemma.

Lemma 1. *Let $z = (\kappa, w)$ and $z' = (\kappa', w')$ be two labels associated with paths p and p' ending at the same node. Label z dominates label z' is the following conditions hold:*

1. $\kappa \leq \kappa'$,
2. $w \leq w'$,
3. *and at least one inequality is strict.*

A label z is said to be feasible if $w \leq W$. The main differences between the dynamic programming algorithms proposed in [13] and [9] is related to the order in which the labels are processed. In [13] the label with minimum cost is selected at each iteration, while they are selected considering the resource consumption in [9]. The algorithm is summarized in Algorithm 1.

Algorithm 1: Label-setting algorithm

initialization: $S_o = (0, 0)$ and $S_i = \emptyset$ for each $i \in N \setminus \{o\}$

while S *is non-empty* **do**
- select a node i such that $S_i \neq \emptyset$ and $(\kappa, w) \in S_i$;
- remove (κ, w) from S_i;
- **foreach** $(i, j) \in A$ **do**
  - **if** $w + r_{ij} \leq W$ **then**
    - create label $(\kappa', w') = (\kappa + \kappa_{ij}, w + r_{ij})$;
    - **if** (κ', w') *is non-dominated by all labels in* S_j **then**
      - add label (κ', w') to S_j;

We can speed up the above algorithm through several preprocessing techniques. First, we can remove from the graphs nodes and arcs that do not belong to the optimal solution, thus obtaining a reduced graph $G' = (N', A')$. These reductions work in two steps. In a first step, we remove node i from G if $w(p_i^+) + w(p_i^-) > W$ where $w(p_i^+)$ and $w(p_i^-)$ are the least resource consumption

from node o to node i and from node i to node d, respectively. A similar rule is applied for each arc (i, j). It is worth observing that if $w(p_d^+) > W$, then the instance is infeasible. In a second step, we solve the Lagrangian relaxation of the problem to obtain lower bounds on the least cost from node o to node i and from node i to node d, denoted by σ_i^+ and σ_i^-, respectively. The feasible paths computed when solving the Lagrangian relaxation are used to compute also an upper bound, UB, on the optimal solution. These bounds can be used similarly to remove nodes and arcs. In this case, a node i that cannot be part of any optimal solution that satisfies $\sigma_i^+ + \sigma_i^- > UB$. Arc (i, j) can be deleted from G when $\sigma_i^+ + \kappa_{ij} + \sigma_j^- > UB$. The aforementioned network reduction techniques do not affect Algorithm 1. When solving the Lagrangean dual problem with the hull approach in [13], least cost paths are computed. If the associated path to node d is feasible, then it is the optimal solution.

Second, we can use upper and lower bounds to further limit the creation of new labels in the course of the algorithm. The idea is to fathom labels associated with partial paths that cannot lead to feasible and optimal solutions. In particular, a label $(\kappa(p_j), w(p_j))$ is discarded if either $w(p_j) + w(p_j^-) > W$ or $\kappa(p_j) + \sigma_j^- > UB$.

3 Robust algorithms

3.1 Sequence of deterministic problems

One of the major results of [4] shows that combinatorial optimization problems with cost uncertainty and the uncertainty polytope $\mathcal{U}^\Gamma$ can be addressed by solving $n + 1$ deterministic problems (where n is the number of binary variables). References [2,10] have recently shown how the aforementioned result can be extended to a robust *constraint* under the uncertainty polytope $\mathcal{U}^\Gamma$. More specifically, the aforementioned works prove the following.

Theorem 1. *Let $\mathcal{Y} \subseteq \{0,1\}^{|I|}$ be the feasibility set of a combinatorial optimization problem and $\Gamma \in \mathbb{Z}$. Moreover, let $\kappa \in \mathbb{R}^{|I|}$ be a cost vector, $b \in \mathbb{R}$, and r be an uncertain vector taking values in $\mathcal{U}^\Gamma$. Without loss of generality, suppose that indices are ordered such that $\hat{r}_1 \geq \hat{r}_2 \geq \cdots \geq \hat{r}_{|I|}$ and let $\hat{r}_{|I|+1}$ be 0. Then, the optimal solution to*

$$\left\{ \min \sum_{i \in I} \kappa_i y_i \ : \ \sum_{i \in I} r_i y_i \leq b \quad \forall r \in \mathcal{U}^\Gamma, \ y \in \mathcal{Y} \right\}$$

is equal to the optimal solution of $\min_{l \in \{1,\ldots,|I|+1\}} Z^l$ where

$$Z^l = \left\{ \min \sum_{i \in I} \kappa_i y_i \ : \ \sum_{i \in I} \overline{r}_i y_i + \sum_{j=1}^{l} (\hat{r}_j - \hat{r}_l) y_j + \leq b - \Gamma \hat{r}_l, \ y \in \mathcal{Y} \right\},$$

for $l = 1, \ldots, |I| + 1$.

We denote by SD the algorithm that solves $RCSPP$ through Theorem 1. Notice that in SD, nodes and arcs are removed from graph $G = (N, A)$ using the aforementioned preprocessing and Algorithm 1 is applied on $G' = (N', A')$ at each iteration.

3.2 Robust label-setting algorithm

Alternatively, problem $\mathcal{U}^\Gamma$-$RCSPP$ can be solved by adapting the label-setting algorithm to the robust situation. In [16], the authors extend the algorithm to $\mathcal{U}^\Gamma$-$RCSPP$ by introducing new robust labels as follows. Given a path $p_j = (i_0 = o, \ldots, i_l = j)$ from o to j, they define the label of the path as

$$\left(\kappa(p), w^0(p_j), \ldots, w^\Gamma(p_j)\right), \tag{1}$$

where $w^\gamma(p_j)$ is defined as the maximum resource consumption along p when considering up to $\gamma \in \{0, \ldots, \Gamma\}$ deviations when $\gamma \leq |p|$ and is equal 0 otherwise, that is,

$$w^\Gamma(p_j) = \begin{cases} \max\limits_{r \in \mathcal{U}^g} \sum\limits_{a \in p_j} r_a, & \text{for each } \gamma \in \{0, \ldots, \min(|p_j|, \Gamma)\}, \\ 0, & \text{for each } \gamma \in \{|p_j| + 1, \ldots, \Gamma\}. \end{cases} \tag{2}$$

Then, we extend the label through arc (j, k), generating a new label for path p_k, with formula

$$\begin{cases} \kappa(p_k) = \kappa(p_j) + \kappa_{jk}, \\ w^0(p_k) = w^0(p_j) + \overline{r}_{jk}, \\ w^\gamma(p_k) = \max\left(w^{\gamma-1}(p_j) + \overline{r}_{jk} + \hat{r}_{jk}, w^\gamma(p_j) + \overline{r}_{jk}\right), \quad \text{for each } \gamma \in \{1, \ldots, \Gamma\}, \end{cases} \tag{3}$$

It is easy to see by induction that extending the label $(0, 0, \ldots, 0)$, that corresponds to the empty path, iteratively through formula (3) leads exactly to definition (2). The worst-case complexity of the resulting algorithm is $O(\Gamma A W^{\Gamma+1})$.

During the course of the label-setting algorithm, labels that will not lead to an optimal solution can be discarded. We consider the cost associated with the label plus the lower bound information derived from the Lagrangian Dual as detailed in the next section. If the new cost is greater than the available upper bound $\bar{\kappa}$, then the label can be fathomed.

A crucial phase in the label-setting algorithm is the removal of dominated labels, which reduces significantly the total number of labels searched in the course of the algorithm. Given two labels y and y' associated to paths p and p' ending at the same node, we say that the label y' is dominated by the label y if the following condition holds: *if path p' belongs to an optimal solution of $RCSPP$, then the path p belongs to an optimal solution of $RCSPP$.*

Dominated labels can be discarded from the search that occurs during the label-setting algorithm. The next result, taken from [16], extends to the robust label the well-known dominance rule.

Lemma 2. *Consider the $\mathcal{U}^\Gamma$-RCSPP and let $z = (\kappa, w^0, \ldots, w^\Gamma)$ and $z' = (\kappa', w'^0, \ldots, w'^\Gamma)$ be two labels associated to paths p and p' ending at the same node. Assume the following conditions hold:*

1. *$\kappa \leq \kappa'$*
2. *$w^j \leq w'^j$, for each $j = 0, \ldots, \Gamma$*
3. *and at least one inequality is strict.*

Then, label z' is dominated by label z.

4 Bounds and Preprocessing for the $\mathcal{U}^\Gamma$-$RCSPP$

In this section, we propose a Lagrangian relaxation for the $\mathcal{U}^\Gamma$-$RCSPP$. The bounds information derived from the resolution of the dual problem are used to perform network reduction procedures and to speed up the search process in the label-setting procedure described in the previous section.

4.1 Lagrangean relaxation

The Lagrangean relaxation is the most effective technique for obtaining valid lower bounds. In addition, the solution process of the Lagrangean dual problem often provides high quality upper bounds. In order to define the Lagrangean relaxation problem, we formulate $\mathcal{U}^\Gamma$-$RCSPP$ as a path-based program below

$$\begin{aligned} \min \quad & \kappa(p) \\ \text{s.t.} \quad & \max_{r \in \mathcal{U}^\Gamma} w(p, r) \leq W \\ & p \in \mathcal{P}. \end{aligned} \tag{4}$$

Taking a Lagrangean relaxation yields the problem $LR(\lambda)$

$$LR(\lambda) = \left\{ \min_{p \in \mathcal{P}} \kappa(p) + \lambda \max_{r \in \mathcal{U}^\Gamma} w(p, r) - \lambda W \right\}, \tag{5}$$

where $\lambda \geq 0$. The solution $Z_{LR}(\lambda)$ of problem (5) represents a lower bound on the solution of (4), for all $\lambda \geq 0$.

For a fixed λ, problem $\overline{LR}(\lambda)$ is a robust version of the Shortest Path Problem (SPP), defined on $\mathcal{U}^\Gamma$, with nominal cost $\bar{\kappa}_a = \kappa_a + \lambda \bar{r}_a$ and $\hat{\kappa}_a = \lambda \hat{r}$. Problem (5) can be solved in polynomial time by solving $|A| + 1$ SPP instances with modified cost or, in alternative, by using the dynamic programming approach proposed in [18]. To find the best lower bound, the Lagrangean dual problem LD must be solved

$$LD = \left\{ \max_{\lambda \geq 0} \overline{LR}(\lambda) \right\}. \tag{6}$$

Problem (6) is solved by iteratively finding an optimal solution to $\overline{LR}(\lambda)$ for different non-negative value of λ. LD can be addressed by several techniques. We mention the cutting plane of Kelley, used in [9], and the hull approach proposed in [13]. It is worth observing that both techniques have polynomial time complexities and solve to optimality problem (6). Actually, this is true for one relaxed constraint. When the Lagrangean problem involves more than one Lagrangean multiplier, the complexity is still an open problem. To overcome this issue, one can solve LD by using the sub-gradient optimization approach (see, e.g. [3]). However, this technique does not guarantee the optimality of the solution.

For each value λ_h, $h = 1, \ldots, H$, where H is the number of $\overline{LR}(\lambda)$ solved, a path p_h from o to d is determined. The cost $\underline{\kappa}(p_h) = -\lambda_h W + \kappa(p_h) + \lambda_h \max_{r \in \mathcal{U}^\Gamma} w(p_h, r)$ is a lower bound on the optimal solution of $\mathcal{U}^\Gamma$-$RCSPP$. In addition, if p_h is feasible, i.e., $\max_{r \in \mathcal{U}^\Gamma} w(p_h, r) \leq W$, then $\overline{\kappa} = \kappa(p_h)$ is a valid upper bound.

We observe that problem $\overline{LR}(\lambda)$ can be solved in order to obtain paths from node d to each other node $j \in N$ with no extra effort. In this respect, for each h, $\gamma = 1, \ldots, \Gamma$, and $j \in N$, we can store the cost $\xi^-_{jh\gamma} = \kappa(p^-_{jh\gamma}) + \lambda_h \max_{r \in \mathcal{U}^\gamma} w(p^-_{jh\gamma}, r)$, where $p^-_{jh\gamma}$ is the optimal path from d to j at iteration h suffering at most γ deviations. As proven by Dumitrescu and Boland [9], the value

$$\sigma^-_{j\gamma}(\mu) = \max_{h=1,\ldots,H} \left(-\lambda_h (W - \mu) + \xi^-_{jh\gamma} \right), \tag{7}$$

is the best lower bound on the optimal path from node j to node d which uses no more resource than $W - \mu$, where $0 \leq \mu \leq W$ is some given value. We call it backward lower bound.

4.2 Network reduction

Resource-based reductions The reduction is based on the computation of lower bounds on the resource consumption. For our application, a valid lower bound is the minimum amount of resource consumption from node o to node i and from node i to node d. Let p^+_i and p^-_i denote the forward and backward paths to node i, respectively. Recall that in the deterministic context (obtained by setting $\hat{r} = 0$), a node i can be removed from the graph if the following condition holds

$$\min_{p^+_i} w(p^+_i, \overline{r}) + \min_{p^-_i} w(p^-_i, \overline{r}) > W. \tag{8}$$

Since any path from o to d crossing node i can be written as $p^-_i \cup p^+_i$, we can rewrite (8) as

$$\min_{p^-_i, p^+_i} w(p^-_i \cup p^+_i, \overline{r}) > W,$$

whose robust counterpart is

$$\min_{p_i^-, p_i^+} \max_{r \in \mathcal{U}^\Gamma} w(p_i^- \cup p_i^+, r) > W. \tag{9}$$

The lhs of (9) can hardly be used for preprocessing because the inner maximization couples the forward and backward paths. We show next how to replace the lhs of (9) by a value that can be expressed directly from the robust forward and backward shortest paths, obtaining a weaker but more easily exploitable removal condition.

$$\begin{aligned}
\min_{p_i^-, p_i^+} \max_{r \in \mathcal{U}^\Gamma} w(p_i^- \cup p_i^+, r) &= \min_{p_i^-, p_i^+} \max_{r \in \mathcal{U}^\Gamma} \big(w(p_i^-, r) + w(p_i^+, r) \big) \\
&= \min_{p_i^-, p_i^+} \max_{\gamma = 0, \dots, \Gamma} \left(\max_{r \in \mathcal{U}^\gamma} w(p_i^-, r) + \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_i^+, r) \right) \\
&\geq \max_{\gamma = 0, \dots, \Gamma} \min_{p_i^-, p_i^+} \left(\max_{r \in \mathcal{U}^\gamma} w(p_i^-, r) + \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_i^+, r) \right) \\
&= \max_{\gamma = 0, \dots, \Gamma} \left(\min_{p_i^-} \max_{r \in \mathcal{U}^\gamma} w(p_i^-, r) + \min_{p_i^+} \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_i^+, r) \right) > W.
\end{aligned} \tag{10}$$

The rhs of (10) involves robust shortest paths from o to i and from i to d, for various values of Γ. These can easily be computed in a preprocessing step and then combined to test the removal of each node i. The same rule can be applied to the arcs. In particular, given an arc (i, j), if the following condition holds

$$\max_{\gamma = 0, \dots, \Gamma} \left(\min_{p_i^+} \max_{r \in \mathcal{U}^\gamma} w(p_i^+, r) + \min_{p_j^-} \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_j^-, r) \right) + \overline{r}_{ij} > W, \tag{11}$$

then, arc (i, j) can be removed.

Cost-based reductions Nodes and arcs can be removed from the network by considering the bound information obtained from the resolution of LD. In particular, letting $\xi^+_{jh\gamma}$ be the least cost from o to j with weight $\kappa(p^+_{jh\gamma}) + \lambda_h \max_{r \in \mathcal{U}^\gamma} w(p^+_{jh\gamma}, r)$, the node j can be removed if the following condition holds

$$\min_{h = 1, \dots, H} \left(\max_{\gamma = 0, \dots, \Gamma} \left(\xi^+_{jh\gamma} + \xi^-_{jh\Gamma - \gamma} \right) - \lambda_h W \right) > \bar{\kappa}. \tag{12}$$

For removing the arcs, we proceed as suggested for the resource-based reductions. In other words, given an arc (i, j), the lhs of condition (12) is modified by considering the cost κ_{ij}. Arc (i, j) is removed if the following condition holds

$$\min_{h = 1, \dots, H} \left(\max_{\gamma = 0, \dots, \Gamma} \left(\xi^+_{ih\gamma} + \xi^-_{jh\Gamma - \gamma} \right) - \lambda_h W \right) + \kappa_{ij} > \bar{\kappa}. \tag{13}$$

4.3 Lower bounds in label-setting algorithm

Label $\left(\kappa(p_j), w^0(p_j), \ldots, w^\Gamma(p_j)\right)$ can be fathomed if the following condition holds

$$\kappa(p_j) + \min_{\gamma=0,\ldots,\Gamma} \sigma^-_{j\Gamma-\gamma}\left(w^\gamma(p_j)\right) > \bar{\kappa}. \tag{14}$$

It is worth observing that we can fathom labels that will not lead to a feasible solution. In particular, if the following condition holds

$$\max_{\gamma=0,\ldots,\Gamma} \left(w^\gamma(p_j) + \min_{p^-_j} \max_{r\in\mathcal{U}^{\Gamma-\gamma}} w(p^-_j, r) \right) > W, \tag{15}$$

then, the label can be discarded for further consideration.

Given a label $\left(\kappa(p_j), w^0(p_j), \ldots, w^\Gamma(p_j)\right)$, we observe that if $\max_{r\in\mathcal{U}^\Gamma} w(p_j \cup p^-_{jh\gamma'}, r) \leq W$ for some $\gamma' \leq \Gamma, h = 1, \ldots, H$, then $Z^j = \kappa(p_j) + \kappa(p^-_{jh\gamma'})$ is an upper bound. Thus, we can improve the upper bound $\bar{\kappa}$ by introducing the following step in the label-setting algorithm

$$\bar{\kappa} = \min\left(\bar{\kappa}, Z^j\right). \tag{16}$$

The value $\max_{r\in\mathcal{U}^\Gamma} w(p_j \cup p^-_{jh\gamma'}, r)$ is computed as $w(p_j \cup p^-_{jh\gamma'}) + \max\{\sum_{a=1}^{\gamma} \hat{r}_a : a \in p_j \cup p^-_{jh\gamma'}, \gamma \leq \Gamma\}$.

The computational results suggest that the gain obtained by better bound in term of generated labels does not suffice the extra effort to compute $\max_{r\in\mathcal{U}^\Gamma} w(p_j \cup p^-_{jh\gamma'}, r)$.

4.4 Robust shortest path problem

The preprocessing routines described in the previous subsections amount to solve many instances of the robust shortest path problem ($\mathcal{U}^\Gamma$-SPP). For the Lagrangian relaxation depicted in Section 4.1, we need to solve $\mathcal{U}^\Gamma$-SPP for different objective functions (one for each dual value λ). Let $\mathcal{P}$ be the set of all binary vectors in $\{0,1\}^{|A|}$ that describe paths from o to d. $\mathcal{U}^\Gamma$-SPP can be cast as

$$\min_{x\in\mathcal{P}} \max_{r\in\mathcal{U}^\Gamma} \sum_{a\in A} r_a x_a.$$

Two approaches have been proposed in the literature to solve $\mathcal{U}^\Gamma$-SPP. The first one, due to [4], is based on the original version of Theorem 1 devised for min max robust combinatorial optimization problems as $\mathcal{U}^\Gamma$-SPP. As Theorem 1, [4] address $\mathcal{U}^\Gamma$-SPP by solving $O(A)$ deterministic versions of the problem. Using Dijkstra's algorithm, the resulting complexity is $O(AN^2)$, which can be decreased to $O(A^2 + AN\log N)$ using a min-priority queue. In this paper, we used instead the extension of the Bellman-Ford algorithm proposed in [18] to solve the problem in $O(\Gamma N^3)$.

In Section 4.2, we need to compute the values of the robust shortest paths to each node, using each value of $0 \leq \gamma \leq \Gamma$. The most efficient way to do that

is again the algorithm proposed in [18], which provides these values at no extra computational cost.

4.5 Outline of the proposed solution strategy

The algorithm is composed of two phases. In the first phase, named preprocessing, network reduction procedures based on resource consumption and cost are performed. The steps of the preprocessing phase are reported in Algorithm 2

Algorithm 2: Preprocessing algorithm

input : graph $G = (N, A)$
output : reduced graph $G' = (N', A')$

Solve forward $\mathcal{U}^\Gamma$-SPP obtaining least robust resource consumption;

if $\min_{p_d^+} \max_{r \in \mathcal{U}^\Gamma} w(p_d^+, r) > W$ **then**
  STOP. Unfeasible instances;

else
  Solve backward $\mathcal{U}^\Gamma$-SPP obtaining least robust resource consumption;
  Perform nodes and arcs reduction (10), (11);
  Solve forward Lagrangean dual problem;
  if $\max_{r \in \mathcal{U}^\Gamma} w(p_h, r) \leq W$, *with* $\lambda_h = 0$ **then**
    STOP. Optimal solution found;
  else
    Solve backward Lagrangean dual problem;
    Perform nodes and arcs reduction (12), (13);

In the second phase, the improved label-setting procedure defined in [16], see Section 4.3, is run on the reduced network $G' = (N', A')$ in order to close the gap between lower bounds and upper bound derived from the preprocessing phase. The steps of the proposed strategy are reported below.

Step 1. *(Preprocessing)*

- Perform Algorithm 2 with $\mathcal{U}^\Gamma = \emptyset$ (deterministic preprocessing).
- Perform Algorithm 2 (robust preprocessing).

Step 2. *(Gap closing)*

- Perform the label-setting algorithm.

5 Computational results

In this section we evaluate the behaviour of the proposed solution strategy. All algorithms have been coded in Java and the numerical results are carried

out on an Intel(R) Core(TM) i7-4720HQ CPU M620 2.60GHz 8.00 GB RAM machine under Microsoft Windows 8. We have considered instances inspired by the scientific literature, which are detailed in the next section.

5.1 Algorithms

We compare in this section the four following algorithms.

$\mathcal{U}^\Gamma$`LSA` denotes the robust label-setting algorithm described in Section 3.2.

`dp-`$\mathcal{U}^\Gamma$`LSA` denotes the algorithm that starts with the deterministic preprocessing and solves the problem on the reduced graph with $\mathcal{U}^\Gamma$`LSA` and using the bounds obtained from the deterministic preprocessing.

`drp-`$\mathcal{U}^\Gamma$`LSA` performs deterministic and robust preprocessing and then solves the problem with $\mathcal{U}^\Gamma$`LSA`, as depicted in Section 4.5.

`SD` solves the problem using the iterative algorithm recalled in Theorem 1. Specifically, `SD` performs first a deterministic preprocessing step, which reduces the number of arcs of the network and hence, the number of iterations in Theorem 1. Then, each deterministic problem involved in Theorem 1 is solved using the classical preprocessing and label-setting algorithm.

5.2 Instances

We have considered four different values of Γ in our experiments, namely $\Gamma \in \{6, 3, 2, 1\}$. For each value of Γ, we have generated instances based on the networks from [3], named b, and a subset of the grid networks from Class 6 used in [9], referred to as G. Specifically, we have restricted ourselves to the 24 instances from [3] with 1 resource only, whose details are summarized in Table 1. For the instances G from [9], we have selected those with numbers of nodes equal to $625, 2005, 5625, 15625$, and numbers of arcs equal to $2400, 9800, 22200, 62000$, respectively, referred to as $G1, G2, G3, G4$.

Test	b1	b2	b3	b4	b9	b10	b11	b12	b17	b18	b19	b20
Nodes	100	100	100	100	200	200	200	200	500	500	500	500
Arcs	955	955	959	959	2040	2040	1971	1971	4858	4858	4978	4978
Density (Arcs/Nodes)	9.55	9.55	9.59	9.59	10.20	10.20	9.86	9.86	9.72	9.72	9.96	9.96

Table 1: Characteristics of the networks presented in [3].

For each of the aforementioned network, we have maintained the original cost and set $\overline{r}$ to the original resource consumption. The value of $\hat{r}$ has been computed as $0.5\,\overline{r}$. We have computed W for each value of Γ. Specifically, we have considered a convex combination of $\max_{r \in \mathcal{U}^\Gamma} w(p^\kappa, r)$ and $\max_{r \in \mathcal{U}^\Gamma} w(p^{\overline{r}}, r)$, that is

$$W = \alpha \max_{r \in \mathcal{U}^\Gamma} w(p^\kappa, r) + (1 - \alpha) \max_{r \in \mathcal{U}^\Gamma} w(p^{\overline{r}}, r), \tag{17}$$

where p^{κ} and $p^{\overline{r}}$ are the paths of minimum cost κ and of minimum resource consumption $\overline{r}$, respectively. The higher the value of α, the higher the resource limit W. In our computational results, we have considered $\alpha \in \{0.25, 0.50, 0.75\}$. We highlight that the instances generated with a given value γ' are feasible for each $\Gamma \leq \gamma'$.

5.3 Numerical results

We compare below the four approaches on the two types of instances.

Networks G. Table 2 shows the average execution times for networks G for each value of Γ and α, including the times spent in preprocessing. The superscript near the execution time under column $\mathcal{U}^{\Gamma}$LSA is the number of instances for which the code runs out of memory.

α	Γ	$\mathcal{U}^{\Gamma}$LSA	dp-$\mathcal{U}^{\Gamma}$LSA	drp-$\mathcal{U}^{\Gamma}$LSA	SD
0.25	6	33.95^{2}	377.03	81.49	227.48
	3	117.42^{1}	150.27	31.44	186.95
	2	106.78^{1}	62.96	27.24	159.93
	1	98.67^{1}	36.01	22.55	112.81
AVG		89.21	156.57	40.68	171.79
0.5	6	32.39^{2}	43.09	44.62	120.42
	3	151.31^{1}	33.51	32.98	82.35
	2	113.98^{1}	32.02	31.71	74.95
	1	106.56^{1}	22.23	24.55	64.00
AVG		101.04	32.71	33.46	85.43
0.75	6	33.55^{2}	33.29	33.55	94.02
	3	118.84^{1}	31.19	27.43	89.41
	2	95.77^{1}	28.17	27.59	55.87
	1	107.22^{1}	23.31	27.39	54.87
AVG		88.85	28.99	28.99	73.54
AVG	6	33.27	151.13	53.22	147.31
	3	129.19	71.66	30.62	119.57
	2	105.51	41.05	28.85	96.92
	1	104.15	27.18	24.83	77.23
AVG		93.03	72.76	34.38	110.26

Table 2: Average computational results for each value of Γ and α on networks G.

On average, the robust preprocessing is useful in terms of computational cost. Indeed, drp-$\mathcal{U}^{\Gamma}$LSA is 2.12 times faster than dp-$\mathcal{U}^{\Gamma}$LSA. In addition, the higher the value of Γ, the higher the speed-up. In particular, drp-$\mathcal{U}^{\Gamma}$LSA is 2.84, 2.34, 1.42, and 1.09 times faster than dp-$\mathcal{U}^{\Gamma}$LSA. For value of α equal to 0.25, the speed-up is 3.85. For $\alpha = 0.50$, dp-$\mathcal{U}^{\Gamma}$LSA is slightly better. Indeed, dp-$\mathcal{U}^{\Gamma}$LSA is 1.02 faster than drp-$\mathcal{U}^{\Gamma}$LSA. For α equal to 0.75, they behave the same.

To better highlight the benefit of using robust preprocessing, Table 3 reports the execution time of the preprocessing and the label-setting algorithm under columns `prep` and `LSA`, respectively. Notice that the time of robust preprocessing contains the time spent in deterministic preprocessing. Column #L reports the number of generated labels. Column #LW shows the number of labels fathomed with the bounds on the resource consumption. Column #LUB reports the number of labels discarded by using bounds on the cost.

	Γ	`prep`	`LSA`	#L	#LW	#LUB
Robust bounds	6	37.44	15.78	46704.33	13412.75	29900.92
	3	28.02	2.60	9357.67	2309.58	15636.75
	2	26.69	2.16	7389.50	1937.00	12123.33
	1	23.02	1.81	5942.42	1563.75	9405.92
AVG		28.79	5.59	17348.48	4805.77	16766.73
Deterministic bounds	6	26.32	124.81	149901.00	29855.83	61110.50
	3	23.38	48.28	65502.75	18698.33	60392.33
	2	22.71	18.34	37170.83	12454.25	37131.17
	1	18.73	8.45	19752.08	6531.92	27375.08
AVG		22.79	49.97	68081.67	16885.08	46502.27

Table 3: Average computational results for each value of Γ, considering deterministic and robust preprocessing separately.

Comparing the results of Table 3, the benefit of using robust information strikes out. Indeed, the execution time of `dp-`$\mathcal{U}^{\Gamma}$`LSA` is 8.95 times slower than `drp-`$\mathcal{U}^{\Gamma}$`LSA`. This behaviour is justified by the number of generated labels. In particular, the label-setting with deterministic bounds generates 3.97 times higher labels than the labeling algorithm with robust bounds. The same trend is observed for #LW and #LUB. The full robust preprocessing is 1.26 times slower than the deterministic one, which means that performing only the robust preprocessing on the graph reduced through deterministic preprocessing is much faster than the deterministic preprocessing itself.

In order to better explain the behaviour of the label-setting algorithm without preprocessing, in Table 4 we report the average execution time for each network varying the value of Γ when the preprocessing in not applied. The entries **M** mean that the code runs out of memory.

The results of Table 4 highlight that the label-setting without preprocessing is not effective in solving the network G. Indeed, the instances generated from network $G4$ and that with Γ equal to 6 from network $G3$ are not solved. Considering the instances solved by $\mathcal{U}^{\Gamma}$`LSA`, `drp-`$\mathcal{U}^{\Gamma}$`LSA` is, on average, 21.54 times faster. The speed-up increases when the Γ increases. In addition, the benefit of robust bounds is more evident for higher dimension networks. Indeed, $\mathcal{U}^{\Gamma}$`LSA` is 1.16, 15.02, and 23.64 times slower than `drp-`$\mathcal{U}^{\Gamma}$`LSA` for network $G1$, $G2$, and $G3$, respectively.

Table 2 highlights that `SD` is slower than `drp-`$\mathcal{U}^{\Gamma}$`LSA`. Indeed, the latter is 3.21 times faster than the former. The speed-up remain almost the same varying Γ. It increases for α equal to 0.25. In this case, `SD` is 4.42 slower than

Γ	$G1$	$G2$	$G3$	$G4$
6	0.69	65.84	**M**	**M**
3	0.58	40.06	346.93	**M**
2	0.45	31.85	284.23	**M**
1	0.44	24.92	287.09	**M**
AVG	0.54	40.67	306.08	

Table 4: Average computational results of the label-setting algorithm without preprocessing for each network, varying Γ.

the proposed approach. For α equal to 0.50 and 0.75 the speed-up is 2.55 and 2.54, respectively.

α	Γ	$\mathcal{U}^\Gamma$LSA	dp-$\mathcal{U}^\Gamma$LSA	drp-$\mathcal{U}^\Gamma$LSA	SD
0.25	6	0.219	0.066	0.107	0.165
	3	0.186	0.077	0.098	0.362
	2	0.168	0.068	0.090	0.427
	1	0.155	0.057	0.070	0.454
AVG		0.182	0.067	0.091	0.352
0.5	6	0.257	0.078	0.120	0.868
	3	0.214	0.066	0.095	1.396
	2	0.202	0.059	0.086	1.779
	1	0.181	0.056	0.072	0.336
AVG		0.213	0.065	0.093	1.095
0.75	6	0.298	0.076	0.118	1.388
	3	0.249	0.066	0.094	2.217
	2	0.238	0.066	0.086	2.693
	1	0.219	0.060	0.079	3.208
AVG		0.251	0.067	0.094	2.377
AVG	6	0.258	0.073	0.115	0.807
	3	0.216	0.070	0.095	1.325
	2	0.203	0.064	0.087	1.633
	1	0.185	0.058	0.074	1.333
AVG		0.215	0.066	0.093	1.275

Table 5: Average computational results for each value of Γ and α on networks b.

Networks b**.** Table 5 shows average results for each value of Γ varying α. The results highlight that dp-$\mathcal{U}^\Gamma$LSA outperforms slightly drp-$\mathcal{U}^\Gamma$LSA. In order to understand this behaviour, Table 6 reports the execution time of the preprocessing (prep) and the label-setting algorithm (LSA), the number of generate labels (#L), the number of labels fathomed with resource bounds (#LW), and the number of labels fathomed with cost bounds (#LUB) when deterministic and robust bounds are considered, respectively.

The label-setting algorithm with robust preprocessing is faster than that

	Γ	prep	LSA	#L	#LW	#LUB
Robust bounds	6	0.112	0.003	98.56	55.44	32.67
	3	0.094	0.001	54.44	75.22	8.67
	2	0.085	0.003	94.22	111.56	56.78
	1	0.072	0.001	47.22	31.72	62.06
AVG		0.091	0.002	73.61	68.49	40.04
Deterministic bounds	6	0.065	0.008	241.94	297.22	34.11
	3	0.060	0.010	186.61	388.22	73.28
	2	0.052	0.013	162.22	413.44	92.94
	1	0.054	0.004	113.06	159.00	90.50
AVG		0.058	0.009	175.96	314.47	72.71

Table 6: Average computational results for each value of Γ, considering deterministic and robust preprocessing separately.

with the deterministic one. Indeed, the latter is 4.39 times slower than the former. This behaviour is justified by the number of generated labels. In particular, #L with deterministic bounds is 2.39 time higher than #L with robust bound. The same trend is observed for #LW and #LUB. However, the benefit of robust bounds in label-setting algorithm does not suffice the extra effort in the preprocessing phase. Indeed, `prep` in the deterministic case is 1.58 times lower than `prep` when robust bounds are computed. The main result is that the approach with deterministic bounds is 1.40 times faster than that with robust ones.

Comparing the average results of column `drp-`$\mathcal{U}^\Gamma$`LSA` and $\mathcal{U}^\Gamma$`LSA` of Table 5, it is observed that the approach with preprocessing is 4.55 times faster than the label-setting algorithm without bound information. `SD` is, on average, 13.72 times slower than `drp-`$\mathcal{U}^\Gamma$`LSA`. The lower the value of Γ, the higher the speed-up. Indeed, the proposed approach is 7.02, 13.88, 18.72, and 18.06 times faster than `SD` for Γ equal to 6, 3, 2, and 1, respectively. A strong relation is observed with the value of α. In particular, the higher α, the slower `SD`. Our approach is 3.86, 11.76, and 25.18 times faster than `SD` for α equal to 0.25, 0.50, and 0.75, respectively.

6 Conclusions

We address in this paper a variant of the resource constrained shortest path problem where the resources consumptions are uncertain parameters. We consider the budgeted uncertainty set introduced by Bertsimas and Sim [4]. Being weakly $\mathcal{NP}$-hard, the problem can be solved in pseudo-polynomial time using a label-setting algorithm [16]. We study here the effect of incorporating a preprocessing phase is the solution algorithm. The preprocessing generalizes the classical deterministic preprocessing, by computing valid lower and upper bounds on the optimal cost and to reduce the original network and removing nodes and arcs that cannot be part of any feasible and/or optimal solutions.

The proposed strategy is tested on instances inspired from the scientific literature. In particular, we have considered the benchmarks proposed in [3] and [9] for the $RCSPP$. We have generated several robust instances by considering different degree of risk of the decision maker.

The computational results shows a good behaviour of the preprocessing phase. Indeed, the computed lower and upper bounds allow to remove up to 92% and 99% of nodes and arcs, respectively. The information derived from the preprocessing phase has an high impact on the gap closing procedure. Indeed, the label-setting algorithm remarkably outperforms the algorithm implementing the well-known iterative algorithm proposed by Bertsimas and Sim in [4] and previously used in [12, 14].

References

[1] A. Agra, M. Christiansen, R. Figueiredo, L. M. Hvattum, M. Poss, and C. Requejo. The robust vehicle routing problem with time windows. *Computers & Operations Research*, 40(3):856 – 866, 2013.

[2] E. Álvarez-Miranda, I. Ljubić, and P. Toth. A note on the bertsimas & sim algorithm for robust combinatorial optimization problems. *4OR*, 11(4):349–360, 2013.

[3] J. E. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

[4] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98:49–71, 2003.

[5] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52:35–53, 2004.

[6] C. Büsing, A. M. C. A. Koster, and M. Kutschka. Recoverable robust knapsacks: Γ-scenarios. In *Network Optimization - 5th International Conference, INOC 2011, Hamburg, Germany, June 13-16, 2011. Proceedings*, pages 583–588, 2011.

[7] W. M. Carlyle, J. O. Royset, and R. K. Wood. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks*, 52(4):256–270, 2008.

[8] L. Di Puglia Pugliese and F. Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

[9] I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42:135 – 153, 2003.

[10] K.-S. Goetzmann, S. Stiller, and C. Telha. Optimization over integers with robustness in cost and few constraints. In *WAOA*, pages 89–101, 2011.

[11] O. Klopfenstein and D. Nace. A robust approach to the chance-constrained knapsack problem. *Oper. Res. Lett.*, 36(5):628–632, 2008.

[12] D. Lu and F. Gzara. The robust crew pairing problem: model and solution methodology. *Journal of Global Optimization*, 62(1):29–54, 2015.

[13] K. Mehlhorn and M. Ziegelmann. Resource constraint shortest paths. In 8^{th} *Ann Eur Symp on Algorithms (ESA2000), LNCS 1879*, pages 326–337, 2000.

[14] S. Mokarami and S. M. Hashemi. Constrained shortest path with uncertain transit times. *Journal of Global Optimization*, 63(1):149–163, 2015.

[15] M. Monaci, U. Pferschy, and P. Serafini. Exact solution of the robust knapsack problem. *Computers & OR*, 40(11):2625–2631, 2013.

[16] A. A. Pessoa, L. D. P. Pugliese, F. Guerriero, and M. Poss. Robust constrained shortest path problems under budgeted uncertainty. *Networks*, 66(2):98–111, 2015.

[17] M. Poss. Robust combinatorial optimization with variable budgeted uncertainty. *4OR*, 11(1):75–92, 2013.

[18] M. Poss. Robust combinatorial optimization with variable cost uncertainty. *European Journal of Operational Research*, 237(3):836 – 845, 2014.

[19] L. D. P. Pugliese and F. Guerriero. A reference point approach for the resource constrained shortest path problems. *Transportation Science*, 47(2):247–265, 2013.

[20] L. Santos, J. Coutinho-Rodrigues, and J. R. Current. An improved solution algorithm for the constrained shortest path problem. *Transport Res B-Meth*, 41(7):756–771, 2007.

[21] M. C. Santos, A. Agra, M. Poss, and D. Nace. A dynamic programming approach for a class of robust optimization problems. Submitted, 2016. http://www.optimization-online.org/DB_HTML/2016/02/5328.html.

A Detailed numerical results

In the following, we give detailed information on the numerical behavior.

A.1 Preprocessing phase evaluation

The subsequent tables show the average results over the considered instances. In particular, columns tw and tξ report the execution time for obtaining lower bounds on the resource consumption and cost, respectively. Columns trw and trξ show the average execution times to perform network reduction based on resource consumption and cost, respectively. Columns %nw (resp %aw) and %nξ (resp %aξ) report the percentage of nodes (resp arcs) reduced using lower bounds on resource consumption and cost, respectively. Columns %n and %a show the average percentages on nodes and arcs removed from the original networks after the network reductions. Column gap reports the average gap between the upper bounds computed during the resolution of the Lagrangean dual problem and the optimal cost. Column #OPT reports the number of instances for which the optimal solution is found when solving the Lagrangean dual problem.

Networks G. Tables 7 shows the average results for network G. As expected, the lower the value of Γ, the lower the execution time. Indeed, the preprocessing procedure for $\Gamma = 1$ is 1.16, 1.22, and 1.63 times faster than the preprocessing for the instances with Γ equal to 2, 3 and 6, respectively. Referring to the effectiveness, no difference in term of percentage of reduced nodes and arcs are shown (see column %n ans %a). However, considering only the resource-based reduction, the preprocessing is more effective for lower value of Γ (see rows AVG of columns %nw and %aw).

Γ	α	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap
	0.25	1.94	1.30	23.37	9.12	7.11%	10.00%	34.48%	37.24%	41.59%	47.24%	9.33%
6	0.50	1.09	0.15	24.04	17.79	3.89%	2.96%	75.75%	81.04%	79.64%	84.00%	0.36%
	0.75	0.59	0.04	14.80	18.10	0.26%	0.46%	90.27%	95.17%	90.53%	95.63%	0.07%
AVG		1.20	0.50	20.73	15.00	3.75%	4.48%	66.83%	71.15%	70.59%	75.63%	3.25%
	0.25	1.16	1.69	13.84	8.19	13.79%	18.89%	27.27%	30.32%	41.05%	49.21%	2.80%
3	0.50	0.68	0.18	15.00	15.89	4.71%	3.58%	74.21%	79.46%	78.93%	83.04%	0.57%
	0.75	0.45	0.04	12.62	14.31	0.31%	0.61%	90.40%	95.23%	90.71%	95.84%	0.04%
AVG		0.76	0.64	13.82	12.80	6.27%	7.69%	63.96%	68.34%	70.23%	76.03%	1.14%
	0.25	0.93	1.87	11.11	8.08	15.84%	22.29%	25.82%	27.79%	41.66%	50.08%	2.30%
2	0.50	0.61	0.22	12.45	17.21	5.02%	4.02%	73.18%	78.54%	78.20%	82.56%	0.83%
	0.75	0.44	0.05	10.56	16.54	0.60%	0.53%	90.43%	95.44%	91.03%	95.98%	0.04%
AVG		0.66	0.71	11.38	13.94	7.15%	8.95%	63.14%	67.26%	70.29%	76.20%	1.06%
	0.25	0.69	2.05	7.70	7.72	19.79%	27.24%	22.43%	23.37%	42.22%	50.61%	1.82%
1	0.50	0.54	0.25	9.49	13.24	6.21%	5.76%	72.40%	76.93%	78.61%	82.69%	0.83%
	0.75	0.41	0.06	10.07	16.85	0.67%	0.69%	90.36%	95.30%	91.03%	95.99%	0.04%
AVG		0.54	0.79	9.09	12.60	8.89%	11.23%	61.73%	65.20%	70.62%	76.43%	0.90%

Table 7: Average computational results of the preprocessing phase varying the parameters Γ and α for networks G.

A substantial differences in terms of nodes and arcs reduction is observed varying the value of α. Considering the resource-reduction procedure, the higher α, the lower the percentage of nodes and arcs reduced (see columns %nw and %aw). This is an expected trend. Indeed, the least resource consumption is the same for each value of α but W increases when α increases. An inverted trend is observed for cost-based reduction. In that case, the lower bound decreases

when α increases, but the quality of the upper bound suffices the worsening in the lower bounds. Indeed, higher quality upper bounds are obtained for higher values of α (see column gap).

deterministic			robust
tw	%n	%a	tw
0.38	26%	28%	0.80
0.39	71%	74%	0.34
0.39	91%	94%	0.08

Table 8: Average computational results of the deterministic and robust preprocessing phase varying the parameter α for networks G.

This behaviour justifies the trend of the execution time for determining lower bounds on resource consumption. Indeed, the higher α, the lower the computational effort (see column tw of Table 7). In Table 8 we show the average results for the deterministic and robust network reduction separately. The execution time for computing lower bounds on the resource consumption during the deterministic preprocessing (first column) is almost the same for each value of α. The percentage of nodes and arcs removed in the deterministic preprocessing raises with α, thus in the robust preprocessing, smaller size networks are considered for higher value of α. The last column shows the execution time for determining the robust lower bounds on the resource consumption. One can observe that the computational effort reduces for higher values of α.

Table 9 shows the behaviour of the preprocessing procedure for each value of Γ varying γ.

Γ	γ	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap
6	6	1.20	0.50	20.73	15.00	3.75%	4.48%	66.83%	71.15%	70.59%	75.63%	3.25%
	3	0.81	0.23	17.42	13.06	0.85%	0.89%	67.07%	71.39%	67.92%	72.28%	1.81%
	2	0.68	0.17	15.57	14.50	0.61%	0.62%	67.18%	71.52%	67.79%	72.15%	1.48%
	1	0.55	0.14	14.21	13.83	0.40%	0.42%	67.19%	71.53%	67.59%	71.95%	1.47%
3	3	0.78	0.62	14.85	12.96	6.27%	7.69%	63.96%	68.34%	70.23%	76.03%	1.14%
	2	0.64	0.53	13.07	13.52	3.38%	3.65%	63.10%	68.09%	66.48%	71.74%	1.68%
	1	0.62	0.48	11.80	14.11	2.06%	2.10%	63.09%	68.12%	65.14%	70.22%	1.52%
2	2	0.64	0.72	11.68	12.89	7.15%	8.95%	63.14%	67.26%	70.29%	76.20%	1.06%
	1	0.53	0.56	10.16	10.84	3.93%	4.67%	62.57%	67.17%	66.50%	71.84%	1.26%

Table 9: Average computational results of the preprocessing phase varying Γ for each value of γ for networks G.

The bounds computed with lower values of γ are less likely to remove nodes and arcs (see columns %n and %a). However a gain in term of computational effort is observed as reported in column tw and tξ.

Networks b**.** Table 10 shows the goodness of preprocessing on networks b. Indeed, the 51% of the instances are solved to optimality. In particular, the lower the value of Γ, the higher the number of instances the preprocessing certificates the optimality. The execution time is limited and the procedure is more efficient for lower value of Γ.

Γ	α	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap	#OPT
6	0.25	0.02	0.03	0.03	0.03	44.47%	34.55%	39.20%	61.99%	83.67%	96.54%	27.27%	5
	0.50	0.03	0.02	0.04	0.03	29.42%	22.91%	44.62%	70.82%	74.03%	93.73%	26.01%	4
	0.75	0.02	0.01	0.04	0.04	18.28%	13.78%	52.50%	77.74%	70.78%	91.52%	35.85%	4
AVG		0.02	0.02	0.03	0.03	30.72%	23.75%	45.44%	70.18%	76.16%	93.93%	29.71%	13
3	0.25	0.02	0.02	0.03	0.03	31.98%	36.10%	51.27%	58.73%	83.25%	94.84%	2.38%	7
	0.50	0.02	0.01	0.04	0.02	26.18%	21.95%	55.93%	74.55%	82.12%	96.50%	2.08%	5
	0.75	0.02	0.01	0.03	0.03	14.15%	12.55%	66.27%	83.76%	80.42%	96.31%	0.00%	7
AVG		0.02	0.01	0.03	0.03	24.11%	23.53%	57.82%	72.35%	81.93%	95.88%	1.49%	19
2	0.25	0.01	0.02	0.03	0.02	26.12%	44.03%	39.53%	45.22%	65.65%	89.26%	9.07%	6
	0.50	0.02	0.02	0.02	0.03	28.80%	25.44%	55.72%	71.96%	84.52%	97.40%	0.00%	6
	0.75	0.02	0.01	0.03	0.03	15.05%	16.40%	53.93%	72.46%	68.98%	88.87%	14.58%	7
AVG		0.01	0.02	0.03	0.02	23.32%	28.62%	49.73%	63.21%	73.05%	91.84%	7.88%	19
1	0.25	0.01	0.02	0.02	0.03	44.87%	46.09%	47.93%	53.04%	92.80%	99.14%	0.00%	8
	0.50	0.01	0.02	0.02	0.02	37.50%	29.63%	47.60%	67.91%	85.10%	97.54%	0.00%	6
	0.75	0.01	0.01	0.02	0.03	23.30%	21.01%	45.65%	68.62%	68.95%	89.63%	14.58%	8
AVG		0.01	0.02	0.02	0.02	35.22%	32.24%	47.06%	63.19%	82.28%	95.44%	4.86%	22

Table 10: Average computational results of the preprocessing phase varying the parameters Γ and α for networks b.

The execution time is not affected by the value of W. However, for higher value of W, we observe a lower percentage of both nodes and arcs removed with lower bounds on resource consumption (see columns %nw and %aw). This is an expected trend. Indeed, the lower bounds are the same for each value of W, but a higher number of feasible paths is present for higher value of W.

An inverted trend is observed for the network reduction based on cost. This behaviour is justified by considering the fact that being the network after resource reductions bigger for higher value of W, a greater numbers of nodes and arcs are removed during the cost-based reductions. However, the overall effectiveness of the network reductions is reduced for high value of W as shown in rows AVG of columns %n and %a. We remark that the quality of the lower bounds, derived from the resolution of the Lagrangean dual problem, decreases for high values of W. Indeed, given a multiplier λ, the cost of the Lagrangean problem is decreased by the constant λW.

Γ	γ	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap
6	6	0.02	0.02	0.03	0.03	30.72%	23.75%	45.44%	70.18%	76.16%	93.93%	29.71%
	3	0.01	0.01	0.02	0.02	24.66%	21.04%	51.71%	73.21%	76.37%	94.25%	6.57%
	2	0.01	0.00	0.02	0.02	19.84%	18.78%	49.83%	71.81%	69.67%	90.59%	6.57%
	1	0.01	0.00	0.01	0.02	14.02%	15.49%	51.56%	71.80%	65.58%	87.29%	19.65%
3	3	0.01	0.01	0.02	0.02	24.11%	23.53%	57.82%	72.35%	81.93%	95.88%	1.49%
	2	0.01	0.01	0.02	0.02	20.11%	21.06%	50.38%	68.51%	70.49%	89.57%	8.15%
	1	0.01	0.00	0.02	0.02	13.65%	16.60%	49.89%	68.17%	63.54%	84.77%	7.68%
2	2	0.01	0.01	0.02	0.02	23.32%	28.62%	49.73%	63.21%	73.05%	91.84%	7.88%
	1	0.01	0.01	0.02	0.02	16.41%	22.81%	46.36%	62.18%	62.77%	84.99%	8.20%

Table 11: Average computational results of the preprocessing phase varying Γ for each value of γ for network b.

In Table 11, we show the performance of the preprocessing on instances with a given value of Γ considering lower bounds computed for $\gamma \leq \Gamma$.

As expected, for each value of Γ, the percentage of removed nodes and arcs decreases for lower values of γ. The execution time is not strongly affected by the different value of γ due to the limited computational effort and to the

dimension of networks b.

A.2 Gap closing phase evaluation

Tables 12–15 show average results. The execution time of the preprocessing and that of the label-setting procedure are reported under column `prep` and `LSA`, respectively. Column #L reports the number of generated labels. Column #LW shows the number of labels fathomed with the bounds on the resource consumption. Column #LUB reports the number of labels discarded by using bounds on the cost.

Networks G. Table 12 shows average results on instances derived from network G. As expected, the higher Γ, the higher the execution time (see column `LSA`). This behaviour is justified by the number of generated labels. Indeed, #L is 7.86, 1.57, and 1.24 times higher for Γ equal to 6, 3, and 2 than the number of labels generated for the instances with $\Gamma = 1$, respectively. The same trend is observed for #LW and #LUB.

Γ	`prep`	`LSA`	#L	#LW	#LUB
6	37.44	15.78	46704.33	13412.75	29900.92
3	28.02	2.60	9357.67	2309.58	15636.75
2	26.69	2.16	7389.50	1937.00	12123.33
1	23.02	1.81	5942.42	1563.75	9405.92
AVG	28.79	5.59	17348.48	4805.77	16766.73

Table 12: Average computational results of the gap closing phase varying Γ for networks G.

The average results, varying the parameter α, are reported in Table 13. Considering the instances with α equal to 0.75 and 0.50, the labeling algorithm is 2644.00 and 12.29 times faster than when solving instances with $\alpha = 0.25$, respectively. This is justified by the number of generated labels. Indeed, #L for α equal to 0.25 and 0.50 is 180.75 and 18.79 times higher than #L for $\alpha = 0.75$. In addition, the number of fathomed labels, i.e. #LW+#LUB, for α equal to 0.25 and 0.50 is 376.51 and 64.63 times higher than that with $\alpha = 0.75$. This justifies the computational overhead for $\alpha = 0.25$. Indeed, a higher number of labels has to be managed.

α	`prep`	`LSA`	#L	#LW	#LUB
0.25	25.19	15.49	44031.13	14058.00	41053.00
0.50	32.20	1.26	7609.44	324.19	9135.94
0.75	28.98	0.01	404.88	35.13	111.25
AVG	28.79	5.59	17348.48	4805.77	16766.73

Table 13: Average computational results of the gap closing phase varying α for networks G.

Network b. The labeling algorithm is very fast in solving the instances derived from network b. Table 14 shows the average results of the label-setting algorithm after the preprocessing. Considering the percentage of #L (%L) over the total number of labels, i.e. #L+#LW+#LUB, it is observed a decreasing trend for lower value of Γ. Indeed, %L is 53%, 39%, 36%, and 33% for Γ equal to 6, 3, 2, and 1, respectively. The fathoming rule based on resource bound is more effective than that based on cost bound. Indeed, the average #LW is 1.71 times higher than #LUB. The only exception is observed for $\Gamma = 1$ where #LUB is 1.96 higher than #LUB.

Γ	prep	LSA	#L	#LW	#LUB
6	0.112	0.003	98.56	55.44	32.67
3	0.094	0.001	54.44	75.22	8.67
2	0.085	0.003	94.22	111.56	56.78
1	0.072	0.001	47.22	31.72	62.06
AVG	0.091	0.002	73.61	68.49	40.04

Table 14: Average computational results of the gap closing phase varying Γ for networks b.

Table 15 shows average results varying the value of α. The higher α, the higher #L. Similar values are observed for α equal to 0.25 and 0.50. For $\alpha = 0.75$ #L grows to 107.96. In particular, #L is 1.83 times higher than #L for $\alpha = 0.25$. The percentage of #LW over the total number of labels (%LW) increases for lower values of α. Indeed, %LW is 61%, 37%, and 24% for α equal to 0.25, 0.50, and 0.75, respectively. An inverted trend is observed for %LUB. In particular, %LUB is 2%, 17%, and 36% for α equal to 0.25, 0.50, and 0.75, respectively.

α	prep	LSA	#L	#LW	#LUB
0.25	0.090	0.001	58.96	96.75	3.33
0.50	0.091	0.002	53.92	42.17	19.25
0.75	0.092	0.003	107.96	66.54	97.54
AVG	0.091	0.002	73.61	68.49	40.04

Table 15: Average computational results of the gap closing phase varying α for networks b.