The resource constrained shortest path problem with uncertain data: a robust formulation and optimal solution approach

Luigi Di Puglia Pugliese, Francesca Guerriero, Michael Poss

The resource constrained shortest path problem with uncertain data: a robust formulation and optimal solution approach

Luigi Di Puglia Pugliese* and Francesca Guerriero†

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

Michael Poss‡

LIRMM, Université de Montpellier, CNRS, Montpellier, France

Abstract

The Resource Constrained Shortest Path Problem ($RCSPP$) models several applications in the fields of transportation and communications. The classical problem supposes that the resource consumptions and the costs are certain and looks for the cheapest feasible path. These parameters are however hardly known with precision in real applications, so that the deterministic solution is likely to be infeasible or suboptimal. We address this issue by considering a robust counterpart of the $RCSPP$. We focus here on resource variation and model its variability through the uncertainty set defined by Bertismas and Sim (2003,2004), which can model the risk aversion of the decision maker through a budget of uncertainty. We solve the resulting problem to optimality through the well-known three phase approach dealing with bounds computation, network reduction and gap closing. In particular, we compute robust bounds on the resource consumption and cost by solving the robust shortest path problem and the dual robust Lagrangian relaxation, respectively. Dynamic programming is used to close the duality gap. Upper and lower bounds are used to reduce the dimension of the network and incorporated in the dynamic programming in order to fathom unpromising states. An extensive computational phase is carried out in order to asses the behavior of the defined strategy comparing its performance with the state-of-the-art. The results highlight the effectiveness of our approach in solving to optimality

*luigi.dipugliapugliese@unical.it, *Corresponding author*

†francesca.guerriero@unical.it

‡michael.poss@lirmm.fr

benchmark instances for $RCSPP$ when Γ is not too large, tailored for the robust counterpart. For larger values of Γ, we show that the most efficient method combines deterministic preprocesing with the iterative algorithm from Bertsimas and Sim (2003). We also illustrate the failure probability of the robust solutions through Monte Carlo sampling.

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 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 is defined as an ordered sequence of l arcs, i.e., $p =< (i_0, i_1), \ldots, (i_{l-1}, i_l) >$ such that $(i_{q-1}, i_q) \in A, q = 1, \ldots, l$. 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 $RCSPP$ is a classical problem in combinatorial optimization, with numerous applications in telecommunications and transportation, among others. The $RCSPP$ models, for instance, the problem faced by route guidance systems for emergency vehicles such as ambulances and fire engines where one wishes to minimize the cost while making sure the travel time does not exceed a given threshold. A similar example arises when choosing the cheapest route using public transportation, where the total travel time is often limited (such as routes reaching airports or train stations). Other applications arise in military aircraft management systems [41], railroad management [16], waste water management [12], and in communications networks [40], which all involve uncertainty that need to be handled adequately to avoid over-conservatism or worse, infeasibility.

The resource consumption is typically uncertain in practical applications. For instance, the travel times mentioned in the previous examples strongly depend on traffic conditions. In maritime transportation, bad weather affects the travel time while in terrestrial transportation, these are also affected by vehicle breakdowns and road constructions. The uncertainty could also result from unknown demands that must be picked up along the routes. There exists a large literature dealing with the stochastic versions of the closely related vehicle routing problems, see [34] and the references therein, where the uncertain parameters are modeled by random variables and the optimization models rely on probabilistic constraints or recourse actions. The literature is less abundant when it comes to the stochastic constrained shortest path problem, see for in-

stance [39]. While stochastic models are quite popular, they suffer from two important drawbacks: first, they require an accurate knowledge of the uncertain parameters (either their density function or a set of scenarios); second, they yield very large-scale optimization problems, even for small networks.

An increasingly prominent alternative to stochastic models considers instead random parameters modeled by convex sets, yielding robust optimization problems. Different uncertainty sets have been introduced in the literature, such as ellipsoids (e.g. [4]) or polytopes defined by few knapsack constraints [33]. A particular polytope that has received a lot of attention is the budgeted uncertainty polytope, first proposed in [5, 6], and largely used in the robust combinatorial optimization literature since then. The success of the budgeted uncertainty polytope can be explained by three main reasons. Firstly, the set benefits from profound links with probabilistic constraints, see the initial bounds from [6] and its extension to variable uncertainty in [31]. Secondly, [5] the set very often leads to robust counterparts that are essentially as easy to solve as their deterministic variants. Thirdly, the set requires little information on the uncertain parameters to be defined: only their mean values, their peak values, and an additional parameter, Γ, which models the risk-averseness of the decision maker. This makes the set easy to use in practice when little historical information is available.

Let us define more precisely the budgeted uncertainty set. 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

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

Given a fixed $r \in \mathcal{U}^{\Gamma}$, we define the resource consumption along a path p as $w(p,r)$ and the constraint related to the maximum resource consumption becomes

$$\max_{r \in \mathcal{U}^{\Gamma}} w(p,r) \leq W. \tag{1}$$

We denote the robust $RCSPP$ under budgeted uncertainty set as $\mathcal{U}^{\Gamma}$-$RCSPP$. Notice that we defined $\mathcal{U}^{\Gamma}$ as a discrete set rather than a polytope. One could as well consider the convex hull of $\mathcal{U}^{\Gamma}$, obtained by replacing the binary restrictions on δ with $\delta_a \in [0,1]$ for each $a \in A$ (see for instance [6]). One readily verifies that, because $w(p,r)$ is convex in r (in fact, linear), considering $\mathcal{U}^{\Gamma}$ or its convex hull has no impact on the set of paths that satisfy the robust constraint (1).

Contributions The purpose of this paper is to define a solution approach tailored for $\mathcal{U}^{\Gamma}$-$RCSPP$, by combining the label-setting algorithm proposed in [30] with new robust bounds inspired by the classical bounds for the problem. Specifically, our main contribution is to show how the classical graph reduction algorithms and bounds obtained from the Lagrangian relaxation of the problem (see [11]) can be extended to the robust context. We conduct extensive

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 [26] and [22], as well as with the dynamic programming from [22] and the classical dualization approach. Our experiments also illustrate the robustness of the solution through a Monte Carlo sampling, showing that the budgeted uncertainty model yields more reliable solutions that the deterministic model, while not loosing much in terms of cost.

In this paper we focus on the particular case where only one resource is considered. However, we highlight that the proposed approach can be easily extended to the more general case with multiple resources.

Structure of the paper The paper is organized as follows. Section 2 further clarifies the literature related to the problem studied in this paper. In Section 3, we apply to $\mathcal{U}^{\Gamma}$-$RCSPP$ two generic algorithms from the literature, the first based on dualization [6] and the second on the iterative algorithm from [5]. Sections 4–6 contain the main contributions of the paper. In particular, Section 4 defines robust Lagrangean relaxation along with hints on how to solve the dual counterpart. In addition, network reduction procedures based on robust information are defined. In Section 5, we adapt the label-setting algorithm proposed in [30] for $\mathcal{U}^{\Gamma}$-$TWSPP$ to handle $\mathcal{U}^{\Gamma}$-$RCSPP$, showing how to incorporate lower and upper bounds information derived from the resolution of the dual Lagrangean relaxation to speed up the search process. Section 6 reports computational results on instances inspired by benchmarks for $RCSPP$. In addition to assessing the computational times and graph reductions of our approach, we also present an illustrative example of the robustness of the solutions. Conclusions are given in Section 7. The appendix reports detailed results on the effect of preprocessing.

2 Related work

We review next the main references related to the problem studied in the paper. First, we mention the classical works on $RCSPP$. We move then to the classical results on budgeted uncertainty. Finally, we present the previous work on the $\mathcal{U}^{\Gamma}$-$RCSPP$ and some works on the closely related robust CVRP and shortest path with time windows.

We emphasize that the label-setting algorithm presented in this manuscript is rather a continuation of the work of [30] (and of [19] as we just discovered while writing this paper), than an adaptation of [5]. Indeed, our work builds on known technique for the $RCSPP$; the result from [5] and its improvements are presented for comparison purpose and because of their importance in the scientific literature.

Constrained shortest path In view of its many applications, the scientific literature has paid great attention to $RCSPP$ and 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 $RCSPP$. We cite, [3, 8, 21] for branch-and-bound schemes, [11,25] for dynamic programming approaches, and [9,35,37] for path ranking methods. Lower and upper bounds information, derived from the resolution of the Lagrangean dual relaxation are useful to speed up the basic algorithms. In addition, network reduction techniques have been developed to reduce the dimension of the network eliminating 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 [10].

Robust combinatorial optimization with budgeted uncertainty The seminal work from [5] proposed an iterative algorithm that solves the robust problem by solving a sequence of deterministic problems. Hence, their result implies that the budgeted uncertainty set keeps the tractability of the min max robust counterparts of many combinatorial optimization problems. The iterative algorithm from [5] has been improved in [2, 18, 20] and extended to problems with robust knapsack-like constraints (as $RCSPP$) independently by [2] and [13]. 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, 17, 27, 36]. Other approaches than dynamic programming have also been considered, see for instance the MILP reformulation proposed in [7] for the recoverable robust knapsack problem.

Robust constrained shortest path Problem $\mathcal{U}^{\Gamma}$-$RCSPP$ has been recently investigated in the following four works. In [30], the authors addressed $\mathcal{U}^{\Gamma}$-$RCSPP$ as well as the version with time windows constraints ($\mathcal{U}^{\Gamma}$-$TWSPP$). Applying the approach from [13], they proved that the former problem is weakly $\mathcal{NP}$-hard. In contrast, they showed that the latter problem is strongly $\mathcal{NP}$-hard. They also developed a general label-setting algorithm for $\mathcal{U}^{\Gamma}$-$TWSPP$, based on the definition of new robust labels. The algorithm has an exponential running-time when Γ is part of the input. Numerical results have been carried out for the $\mathcal{U}^{\Gamma}$-$TWSPP$. In [26], the authors study $\mathcal{U}^{\Gamma}$-$RCSPP$ as well as the version where the uncertainty set is an ellipsoid. For the first problem (budgeted uncertainty), they extend the iterative algorithm from [5] to $\mathcal{U}^{\Gamma}$-$RCSPP$. For the second problem (with ellipsoidal uncertainty), they proposed a new iterative algorithm that is based on a parametric reformulation of the problem. They also linked the problem with ellipsoidal uncertainty to the chance-constrained version of $RCSPP$. They presented numerical results for both problems (with budgeted and ellipsoidal uncertainty). In [22,24], the authors proposed a label-setting algorithm and network reduction techniques for $\mathcal{U}^{\Gamma}$-$RCSPP$ that is based on lower bounds derived from the resolution of a modified deterministic Lagrangean relaxation. In addition, the author also adapted the iterative algorithm from [5] to $\mathcal{U}^{\Gamma}$-$RCSPP$ and compared it numerically to the label-setting algorithm. The authors further considered these methods in [23], where a La-

grangian relaxation has been applied to a robust crew pairing problem ending up with a subproblem identical to $\mathcal{U}^{\Gamma}$-$RCSPP$. While writing this paper, we also became aware of the work of [19] where the authors propose a label-setting algorithm similar to the one from [30].

The strategy proposed in this manuscript differs from the work realized in [22, 24]. On the one hand, the label-setting algorithm considered in [22, 24] is based on the deterministic labels but uses a different dominance rule. Showing how the classical dominance cannot be applied in the robust case, the authors handle the uncertain part of the resource consumption by a conservative bound. Similarly, the Lagrangean problem proposed in [22, 24] relies on deterministic shortest path computation. The obtained lower bounds are tightened considering a-priori lower bound on the resource consumption $\hat{r}$. As a consequence, they perform a network reduction that considers nominal values and take into account the robust part as a constant. On the other hand, the proposed modified label-setting algorithm relies on new robust labels whose dimensions depend on Γ, extending to these labels the dominance rule used for problems with multiple resources. Similarly, the Lagrangean problem addressed in the present paper involves solving robust shortest path problems. Said differently, we do not get rid of the robustness by using conservative bounds.

The closely related vehicle routing problem with uncertain demands has also been considered in the robust context. The first works dates back to [38] who studied conditions under which the robust problems can be reformulated deterministically; further work has then considered less restrictive models [28]. Later, the authors of [15] tackled the problem by extending classical compact formulations to the robust context. They further study empirically the connection of the robust problem with the ambiguous counterpart, where the capacity must be satisfied with high probability for all probability distributions in given ambiguity sets. The authors of [14] proposed a metaheuristic based on adaptive memory programming, able to provide good solutions in reasonable amounts of time for many of the instances studied in [15]. More recently, [29] have proposed a branch-and-cut-and-price algorithm able to solve optimally many of these instances.

3 Generic approaches

We recall below two well-known approaches for solving robust combinatorial optimization problems. The two approaches rely on a linear programming formulation for $\mathcal{U}^{\Gamma}$-$RCSPP$. Let $x \in \{0,1\}^m$ be a vector of optimization variables stating which arcs belong to the solution, and let $\mathcal{X} \subset \{0,1\}^m$ contain all vectors x that correspond to paths from o to d. Hence, the set $\mathcal{X}$ is typically defined by flow conservations constraints and binary restrictions on x. With

these variables, $\mathcal{U}^\Gamma$-$RCSPP$ can be cast as

$$\begin{aligned} \min \quad & \sum_{a\in A} \kappa_a x_a \\ \text{s.t.} \quad & \sum_{a\in A} r_a x_a \le W, \quad r \in \mathcal{U}^\Gamma \\ & x \in \mathcal{X}. \end{aligned} \tag{2}$$

3.1 Dualization

Let us recall that constraint (2) is equivalent to constraint

$$\sum_{a\in A} r_a x_a \le W, \quad r \in \text{conv}(\mathcal{U}^\Gamma), \tag{3}$$

where the convex hull of $\mathcal{U}^\Gamma$ can be formulated as the following polytope (see [6] for details)

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

A celebrated result in robust optimization [4] reformulates (3) as

$$\max_{r\in \text{conv}(\mathcal{U}^\Gamma)} \sum_{a\in A} r_a x_a \le W,$$

then dualizes the maximization linear program to obtain

$$\sum_{a\in A} \overline{r}_a x_a + \Gamma z + \sum_{a\in A} y_a \le W, \tag{4}$$

$$z + y_a \ge \hat{r}_a x_a, \quad a \in A, \tag{5}$$

$$z, y \ge 0 \tag{6}$$

where z and y are the variables dual to the constraints defining $\text{conv}(\mathcal{U}^\Gamma)$. The resulting linear integer program contains restrictions (4)–(6) together with $x \in \mathcal{X}$.

3.2 Iterative algorithm

An alternative approach, proposed in [5], shows that a combinatorial optimization problem with cost uncertainty and the uncertainty polytope $\mathcal{U}^\Gamma$ can be addressed by solving $|I| + 1$ deterministic problems, where $|I|$ is the number of binary variables of the optimization problem. References [2,13,20] have recently shown how the aforementioned result can be extended to a robust *constraint* under the uncertainty polytope $\mathcal{U}^\Gamma$, and reduced the number of problems to be solved. 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 $\hat{r}_1 \geq \hat{r}_2 \geq \cdots \geq \hat{r}_{|I|}$, let $\hat{r}_{|I|+1}$ be 0, and define $\mathcal{L} = \{\Gamma+1, \Gamma+3, \Gamma+5, \ldots, \Gamma+\gamma, |I|+1\}$ where γ is the largest odd integer such that $\Gamma + \gamma < |I| + 1$. 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 $\min_{l \in \mathcal{L}} Z^l$ 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 $l \in \mathcal{L}$.

Theorem 1 can be applied to the formulation for $\mathcal{U}^\Gamma$-$RCSPP$ provided above, wich contains only one robust constraint, (2). We denote by `SD` the algorithm that applies Theorem 1 to the above formulation, thus solving $\lceil (|I| - \Gamma)/2 \rceil + 1$ deterministic $RCSPP$.

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

In this Section, we propose a Lagrangian relaxation for $\mathcal{U}^\Gamma$-$RCSPP$. The bounds information derived from the resolution of the dual problem is used to perform network reduction procedures and to speed up the search process in the label-setting procedure described in the Section 5.

4.1 Lagrangean relaxation

The Lagrangean relaxation is an effective technique for obtaining strong lower bounds to complex integer programs. 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 let $\mathcal{P}$ denote the set of all paths from o to d and rewrite problem (2) as

$$\begin{aligned} \min \quad & \kappa(p) \\ \text{s.t.} \quad & \max_{r \in \mathcal{U}^\Gamma} w(p,r) \leq W \\ & p \in \mathcal{P}. \end{aligned} \tag{7}$$

Taking a Lagrangean relaxation for constraint (7) yields the problem $LR(\lambda)$

$$LR(\lambda) = \min_{p \in \mathcal{P}} \left\{ \kappa(p) + \lambda \max_{r \in \mathcal{U}^\Gamma} w(p,r) - \lambda W \right\}, \tag{8}$$

where $\lambda \geq 0$. The solution of problem (8), denoted $Z_{LR}(\lambda)$ hereafter, provides a valid lower bound on the solution of (7), for all $\lambda \geq 0$.

For a fixed λ, problem $LR(\lambda)$ is an instance of the robust shortest path problem with budgeted uncertainty (denoted $\mathcal{U}^{\Gamma}$-SPP), with nominal cost equal to $\overline{c}_a = \kappa_a + \lambda \overline{r}_a$ and deviation equal to $\hat{c}_a = \lambda \hat{r}$. Therefore, problem (8) can be solved in polynomial time using the label-correcting algorithm from [32], having a complexity of $O(\Gamma n^3)$. 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 [32], which provides these values at no extra computational cost.

To find the best lower bound, the Lagrangean dual problem LD must be solved

$$LD = \left\{ \max_{\lambda \geq 0} LR(\lambda) \right\}. \tag{9}$$

Among the different techniques available to solve problem LD, we use here the hull algorithm proposed in [25], which solves the problem to optimality. Notice that when only one constraint is relaxed, the algorithm has a polynomial-time running time; its complexity is open when the Lagrangean problem involves more than one Lagrangean multiplier. However, in the latter case, one could still solve (not to optimality) problem (9) by means of the subgradient method.

Let λ_h denote the sequence of multipliers generated when solving (9), $h = 1, \ldots, H$, where H is the number of problems of type $LR(\lambda)$ that are solved. For each value of λ_h, 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.

Notice that, while solving problem $LR(\lambda)$, we obtain at no extra computational costs, paths from node d to each other node $j \in N$ by reversing the arc directions. 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 (using the reversed arcs) at iteration h suffering at most γ deviations. As proven by [11], the value

$$\sigma^-_{j\gamma}(\mu) = \max_{h=1,\ldots,H} \left(-\lambda_h (W - \mu) + \xi^-_{jh\gamma} \right), \tag{10}$$

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 lower bounds on the resource consumption. Such a valid lower bound is given by 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

$$\min_{p_i^+} w(p_i^+, \overline{r}) + \min_{p_i^-} w(p_i^-, \overline{r}) > W. \tag{11}$$

Since any path from o to d crossing node i can be written as $p_i^- \cup p_i^+$, we can rewrite (11) as

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

whose robust counterpart is

$$\min_{p_i^-, p_i^+} \max_{r \in \mathcal{U}^\Gamma} w(p_i^- \cup p_i^+, r) > W. \tag{12}$$

The left hand side (lhs) of (12) 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 (12) 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.

$$\begin{aligned}
\min_{p_i^-, p_i^+} \max_{r \in \mathcal{U}^\Gamma} w(p_i^- \cup p_i^+, r) &= \min_{p_i^-, p_i^+} \max_{r \in \mathcal{U}^\Gamma} \big(w(p_i^-, r) + w(p_i^+, r) \big) \\
&= \min_{p_i^-, p_i^+} \max_{\gamma = 0, \dots, \Gamma} \left(\max_{r \in \mathcal{U}^\gamma} w(p_i^-, r) + \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_i^+, r) \right) \\
&\geq \max_{\gamma = 0, \dots, \Gamma} \min_{p_i^-, p_i^+} \left(\max_{r \in \mathcal{U}^\gamma} w(p_i^-, r) + \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_i^+, r) \right) \\
&= \max_{\gamma = 0, \dots, \Gamma} \left(\min_{p_i^-} \max_{r \in \mathcal{U}^\gamma} w(p_i^-, r) + \min_{p_i^+} \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_i^+, r) \right) > W.
\end{aligned} \tag{13}$$

The right hand side (rhs) of (13) 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

$$\max_{\gamma = 0, \dots, \Gamma} \left(\min_{p_i^+} \max_{r \in \mathcal{U}^\gamma} w(p_i^+, r) + \min_{p_j^-} \max_{r \in \mathcal{U}^{\Gamma - \gamma}} w(p_j^-, r) \right) + \overline{r}_{ij} > W, \tag{14}$$

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

$$\min_{h=1,\dots,H}\left(\max_{\gamma=0,\dots,\Gamma}\left(\xi^+_{jh\gamma}+\xi^-_{jh\Gamma-\gamma}\right)-\lambda_h W\right) > \bar{\kappa}. \tag{15}$$

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 (15) is modified by considering the cost κ_{ij}. Arc (i,j) is removed if the following condition holds

$$\min_{h=1,\dots,H}\left(\max_{\gamma=0,\dots,\Gamma}\left(\xi^+_{ih\gamma}+\xi^-_{jh\Gamma-\gamma}\right)-\lambda_h W\right)+\kappa_{ij} > \bar{\kappa}. \tag{16}$$

5 Modified label-setting algorithm

Problem $\mathcal{U}^\Gamma$-$RCSPP$ can be solved by adapting the label-setting algorithm proposed in [30] to address the $\mathcal{U}^\Gamma$-$TWSPP$. We recall shortly in Section 5.1 their algorithm, rewritten for problem $\mathcal{U}^\Gamma$-$RCSPP$, and refer the interested reader to [30] for a more detailed description, including a proof of correctness. Section 5.2 then presents the new bounds proposed in the present manuscript. The resulting overall algorithm, tailored for $\mathcal{U}^\Gamma$-$RCSPP$, is presented in Section 5.3.

5.1 Robust labels

Given a path $p_j = (i_0 = o, \dots, i_l = j)$ from o to j, the label of the path is defined as

$$\left(\kappa(p), w^0(p_j), \dots, w^\Gamma(p_j)\right), \tag{17}$$

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

$$w^\gamma(p_j) = \begin{cases} \max\limits_{r\in\mathcal{U}^\gamma} \sum\limits_{a\in p_j} r_a, & \text{for each } \gamma \in \{0,\dots,\min(|p_j|,\Gamma)\}, \\ 0, & \text{for each } \gamma \in \{|p_j|+1,\dots,\Gamma\}. \end{cases} \tag{18}$$

Then, we extend the label through arc (j,k) with the formula

$$\begin{cases} \kappa(p_k) = \kappa(p_j) + \kappa_{jk}, \\ w^0(p_k) = w^0(p_j) + \overline{r}_{jk}, \\ w^\gamma(p_k) = \max\left(w^{\gamma-1}(p_j) + \overline{r}_{jk} + \hat{r}_{jk}, w^\gamma(p_j) + \overline{r}_{jk}\right), & \text{for each } \gamma \in \{1,\dots,\Gamma\}. \end{cases} \tag{19}$$

It is easy to see by induction that extending the label $(0,0,\dots,0)$, that corresponds to the empty path, iteratively through formula (19) leads exactly to definition (18). The worst-case complexity of the resulting algorithm is $O(\Gamma m W^{\Gamma+1})$.

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.

Lemma 1 *[30] Consider $\mathcal{U}^\Gamma$-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. *$w^\gamma \leq w'^\gamma$, for each $\gamma = 0, \ldots, \Gamma$*
3. *and at least one inequality is strict.*

then, label z' is dominated by label z.

Dominated labels can be discarded from the search that occurs during the label-setting algorithm.

5.2 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

$$\kappa(p_j) + \min_{\gamma=0,\ldots,\Gamma} \sigma^-_{j\Gamma-\gamma}\left(w^\gamma(p_j)\right) > \bar{\kappa}. \tag{20}$$

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

$$\max_{\gamma=0,\ldots,\Gamma} \left(w^\gamma(p_j) + \min_{p_j^-} \max_{r \in \mathcal{U}^{\Gamma-\gamma}} w(p_j^-, r) \right) > W, \tag{21}$$

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

$$\bar{\kappa} = \min\left(\bar{\kappa}, Z^j\right). \tag{22}$$

The value $\max_{r\in\mathcal{U}^\Gamma} w(p_j \cup p^-_{jh\gamma'}, r)$ is computed as

$$w(p_j \cup p^-_{jh\gamma'}) + \max\left\{ \sum_{a\in S} \hat{r}_a : S \subseteq p_j \cup p^-_{jh\gamma'}, |S| \leq \Gamma \right\}.$$

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

5.3 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 1. Their computational cost is summarized below.

1. Compute robust shortest paths from o and d to all nodes of the graph, using twice the dynamic programming algorithm suggested in [32] with cost $r_a, \forall a \in A$. The time complexity is $O(\Gamma n^3)$.

2. Solve the Lagrangean relaxation of $\mathcal{U}^\Gamma$-$RCSPP$ using the hull approach and the aforementioned dynamic programming algorithm. Applying the same reasoning as in [25], we obtain a time complexity of $O(\log(nK(\overline{R} + \hat{R})\Gamma n^3)$, where $\overline{R} = \max_{a \in A} \overline{r}_a$ and $\hat{R} = \max_{a \in A} \hat{r}_a$.

Algorithm 1: 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 (13), (14);
  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 (15), (16);

In the second phase, the modified label-setting procedure 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. *(Network reduction)*

- Perform Algorithm 1 with $\Gamma = 0$ (deterministic preprocessing).
- Perform Algorithm 1 (robust preprocessing).

Step 2. *(Solving the problem on the reduced graph)*

- Perform the label-setting algorithm described in Section 5 on the reduced graph, with pruning procedures (20) and (21).

Algorithm 1 is iteratively performed until arcs and nodes are removed in both deterministic and robust setting.

6 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. All computational times are reported in seconds.

6.1 Algorithms

We compare in this Section the seven following algorithms.

$\mathcal{U}^\Gamma$`LSA` denotes the robust label-setting algorithm described in Section 5 that do not use any bounds information.

`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. Specifically, the algorithm corresponds to the one described in Section 5.3, ommitting the second item of **Step 1**.

`drp-`$\mathcal{U}^\Gamma$`LSA` performs deterministic and robust preprocessing and then solves the problem with $\mathcal{U}^\Gamma$`LSA`, as depicted in Section 5.3.

`GD` implements the preprocessing and the label-setting algorithm proposed in [22].

`SD` solves the problem using the iterative algorithm recalled in Theorem 1 from Section 3.2. 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 deterministic preprocessing and label-setting algorithm with $\Gamma = 0$. Notice that, since `SD` amounts to solve a sequence of deterministic problem, *the robust preprocessing cannot be used for `SD`.*

`dual` solves the dualized problem described in Section 3.1.

`dp-dual` solves the dualized problem described in Section 3.1 on the graph obtained after deterministic preprocessing.

The listed algorithms are coded in Java language. In order to carry out a fair comparison among the considered solution strategies, we implemented them by using the same data structure.

6.2 Instances

We have considered seven different values of Γ in our experiments, namely $\Gamma \in \{20, 15, 10, 6, 3, 2, 1\}$. 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 [11], referred to as G. Specifically, we have restricted ourselves to the 12 instances from [3] with 1 resource only, whose details are summarized in Table 1. For the instances G from [11], we have selected those with numbers of nodes equal to $625, 2500, 5625, 15625$, and numbers of arcs equal to $2400, 9800, 22200, 62000$, respectively, referred to as $G1, G2, G3, G4$. Preliminary computational results show that the optimal paths of network b have from 3 to 10 arcs. Thus, in order to generate and test meaningful instances, we consider $\Gamma \in \{6, 3, 2, 1\}$ for network b.

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

$$W = \alpha \max_{r \in \mathcal{U}^\Gamma} w(p^\kappa, r) + (1 - \alpha) \max_{r \in \mathcal{U}^\Gamma} w(p^{\overline{r}}, r), \tag{23}$$

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 $\alpha \in \{0.25, 0.50, 0.75\}$. Notice that W as defined in (23) increases with the value of Γ. Therefore, the instances generated with a given value γ are feasible for each $\Gamma \leq \gamma$.

The computational results for networks G reveal a different behavior of the proposed solution approach with respect to the value of Γ. In particular, for $\Gamma \in \{6, 3, 2, 1\}$, `drp-`$\mathcal{U}^\Gamma$`LSA` is able to solve all the considered instances. Instead, for $\Gamma \in \{20, 15, 10\}$ not all instances are solved to optimality. For this reason, we split the discussion by considering the two sets for the values of Γ seperately.

6.3 Computational efficiency

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

Networks G for $\Gamma \in \{6, 3, 2, 1\}$. 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. The superscript in column GD indicates the number of instances for which the code does not provide a solution within the imposed time limit set to 900 seconds. The rows AVG slv show the execution time averaged on the instances solved by GD.

α	Γ	$\mathcal{U}^\Gamma$LSA	dp-$\mathcal{U}^\Gamma$LSA	drp-$\mathcal{U}^\Gamma$LSA	GD	SD	dual	dp-dual
0.25	6	33.95^2	377.03	81.49	160.39^1	113.42	540.02	420.78
	3	117.42^1	150.27	31.44	87.26^1	122.17	185.84	105.89
	2	106.78^1	62.96	27.24	86.35^1	138.27	113.60	88.93
	1	98.67^1	36.01	22.55	69.09^1	110.72	57.21	58.25
AVG		89.21	156.57	40.68		121.15	224.17	168.46
AVG slv			32.50	20.10	100.77			
0.5	6	32.39^2	43.09	44.62	151.72^1	70.43	239.37	49.59
	3	151.31^1	33.51	32.98	126.05	58.62	99.38	95.97
	2	113.98^1	32.02	31.71	109.63	57.56	117.46	41.76
	1	106.56^1	22.23	24.55	96.98	48.11	119.30	36.21
AVG		101.04	32.71	33.46		58.68	143.88	55.88
AVG slv			23.33	23.73	121.10			
0.75	6	33.55^2	33.29	33.55	27.61^1	59.69	39.34	32.89
	3	118.84^1	31.19	27.43	27.12	60.93	38.22	27.39
	2	95.77^1	28.17	27.59	26.20	49.11	38.71	27.80
	1	107.22^1	23.31	27.39	26.75	47.61	37.59	27.95
AVG		88.85	28.99	28.99		54.33	38.47	29.01
AVG slv			21.66	21.66	26.92			
AVG	6	33.27	151.13	53.22	113.24	81.18	272.91	167.76
	3	129.19	71.66	30.62	80.14	80.57	107.81	76.42
	2	105.51	41.05	28.85	74.06	81.65	89.92	52.83
	1	104.15	27.18	24.83	64.27	68.81	71.37	40.80
AVG		93.03	72.76	34.38		78.05	135.50	84.45
AVG slv			25.83	21.83	82.93			

Table 2: Average computational results for each value of Γ and α on networks G for $\Gamma \in \{6, 3, 2, 1\}$.

On average, the robust preprocessing is useful in terms of computational cost. Indeed, we see from the last row AVG of Table 2 that 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. The table also shows that the generic approaches SD and dual are less efficient than drp-$\mathcal{U}^\Gamma$LSA. Indeed, SD and dp-dual are, on average, 2.27 and 2.46 times slower than drp-$\mathcal{U}^\Gamma$LSA, respectively. When only deterministic preprocessing is applied, however, dp-$\mathcal{U}^\Gamma$LSA is only 1.16 times faster than dp-dual, on average.

The proposed approaches outperform GD, which is not able to provide a solution within the imposed time limit for a total of 6 instances out of 48. The network $G4$ is not solved for all values of Γ when $\alpha = 0.25$ and for α equals to 0.50 and 0.75 and $\Gamma = 6$. dp-$\mathcal{U}^\Gamma$LSA and drp-$\mathcal{U}^\Gamma$LSA are, on average, 3.21 and 3.80 times faster than GD (see last row AVG slv).

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 is not applied. The entries **M** means 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 Γ increases. In addition, the benefit of robust bounds is more evident for higher dimension networks. Indeed, $\mathcal{U}^\Gamma$LSA

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.

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 remains almost the same varying Γ. It increases for α equal to 0.25. In this case, SD is 4.42 slower than the proposed approach. For α equal to 0.50 and 0.75 the speed-up is 2.55 and 2.54, respectively.

Networks G for $\Gamma \in \{20, 15, 10\}$. These instances are more complex so that all algorithms but SD leave some of them unsolved within the time limit of 900 seconds.

Table 5 shows the average execution time for each value of α and Γ. The superscript near the average computational effort indicates the number of instances not solved within the time limit imposed. Table 5 provides the average execution time for each value of α and solution approach over the instances solved by both drp-$\mathcal{U}^\Gamma$LSA and the corresponding algorithm.

The first observation is that drp-$\mathcal{U}^\Gamma$LSA is not able to solve 2 instances out of 4 for α equal to 0.25. This situation occurs for all the three values of Γ. All instances are solved to optimality for α equal to 0.50 and 0.75.

The same behaviour is observed for dp-$\mathcal{U}^\Gamma$LSA for $\alpha = 0.25$. In addition, for $\alpha = 0.50$, the number of instances not solved is equal to 2, 1, and 1 for values of Γ equal to 20, 15, and 10, respectively. Considering only the instances solved by both dp-$\mathcal{U}^\Gamma$LSA and drp-$\mathcal{U}^\Gamma$LSA, the former is 6.26 and 1.46 times slower than drp-$\mathcal{U}^\Gamma$LSA for α equal to 0.25, 0.50, respectively. dp-$\mathcal{U}^\Gamma$LSA is slightly faster than drp-$\mathcal{U}^\Gamma$LSA for $\alpha = 0.75$. Indeed, drp-$\mathcal{U}^\Gamma$LSA is 1.03 times slower than dp-$\mathcal{U}^\Gamma$LSA. This behaviour reveals the effectiveness of the robust preprocessing for more constrained instances.

Overall, the preprocessing procedure improves the performance of the labelling algorithm. Indeed, considering only the instances solved by both drp-$\mathcal{U}^\Gamma$LSA and $\mathcal{U}^\Gamma$LSA, the latter is 12.86, 26.72, and 34.60 times slower than the former for α equal to 0.25, 0.50 and 0.75, respectively.

The GD approach is not able to solve to optimality 2 instances out of 4 for each value of α and Γ. Thus, 18 instances can not be solved by GD against the 6 not solved by drp-$\mathcal{U}^\Gamma$LSA. Considering only the instances solved by both GD and drp-$\mathcal{U}^\Gamma$LSA, the results collected in Table 5, highlight the superiority

of drp-$\mathcal{U}^\Gamma$LSA. Indeed, drp-$\mathcal{U}^\Gamma$LSA is 29.48, 59.10, and 82.06 times faster than GD, for α equal to 0.25, 0.50, and 0.75, respectively.

drp-$\mathcal{U}^\Gamma$LSA is 4.07 times slower than SD (see last row of Table 5). In more details, the higher the value of α the lower the speed up of SD with respect to drp-$\mathcal{U}^\Gamma$LSA. In particular, the former is 4.46, 31.34, and 1.23 times faster than the latter for α equal to 0.25, 0.50, and 0.75, respectively.

Referring to the dualized approaches, both dual and dp-dual are not able to solve 9 instances. As expected, dp-dual is more efficient. drp-$\mathcal{U}^\Gamma$LSA is, on average, 2.07 times faster than dp-dual, considering the instances solved by both approaches. More in details, dp-dual is 12.84 and 5.52 times slower than drp-$\mathcal{U}^\Gamma$LSA for α equal to 0.25 and 0.50, respectively. The trend is inverted for $\alpha = 0.75$, that is, dp-dual is 8.50 times faster than drp-$\mathcal{U}^\Gamma$LSA.

α	Γ	$\mathcal{U}^\Gamma$LSA	dp-$\mathcal{U}^\Gamma$LSA	drp-$\mathcal{U}^\Gamma$LSA	GD	SD	dual	dp-dual
0.25	20	59.18^2	2.65^2	3.33^2	202.18^2	43.59	180.63^2	17.77^2
	15	52.62^2	42.24^2	4.05^2	104.37^2	43.15	99.78^2	84.02^2
	10	43.12^2	30.48^2	4.66^2	48.55^2	65.49	66.76^2	52.91^2
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-$\mathcal{U}^\Gamma$LSA			25.12	4.02				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and $\mathcal{U}^\Gamma$LSA		51.64		4.02				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and GD				4.02	118.36			
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and SD				4.02		0.90		
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dual				4.02			115.72	
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-dual				4.02				51.57
0.5	20	65.73^2	2.91^2	155.74	207.26^2	40.34	160.54^1	134.89^1
	15	50.65^2	13.77^1	85.90	106.93^2	39.84	99.36^1	12.83^1
	10	43.30^2	9.45^1	69.28	39.04^2	59.28	52.25^1	13.58^1
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-$\mathcal{U}^\Gamma$LSA			9.44	6.44				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and $\mathcal{U}^\Gamma$LSA		53.22		1.99				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and GD				1.99	117.74			
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and SD				103.64		3.31		
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dual				9.74			104.05	
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-dual				9.74				53.77
0.75	20	58.88^2	42.14	47.39	211.59^2	24.23	102.96	9.19
	15	50.62^2	41.11	37.64	109.22^2	24.16	56.26	3.22
	10	42.42^2	31.73	33.23	39.48^2	47.52	46.59	1.51
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-$\mathcal{U}^\Gamma$LSA			38.33	39.42				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and $\mathcal{U}^\Gamma$LSA		50.64		1.46				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and GD				1.46	120.10			
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and SD				39.42		31.97		
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dual				39.42			68.60	
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-dual				39.42				4.64
AVG	20	61.26	15.90	17.53	207.01	32.29	131.75	72.04
	15	51.29	32.37	16.21	106.84	32.00	77.81	8.03
	10	42.95	23.88	15.62	42.35	53.40	49.42	7.55
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-$\mathcal{U}^\Gamma$LSA			24.29	16.62				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and $\mathcal{U}^\Gamma$LSA		51.83		2.49				
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and GD				2.49	118.73			
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and SD				49.02		12.06		
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dual				17.72			96.13	
AVG solved by both drp-$\mathcal{U}^\Gamma$LSA and dp-dual				17.72				36.66

Table 5: Average computational results for each value of Γ and α on networks G for $\Gamma \in \{20, 15, 10\}$.

As many of the instances are not solved within the time limit, we give an alternative perspective on the behaviour of the solution approaches considered Figure 1. The later highlights the superiority of SD for these instances. That algorithm is indeed the fastest for roughly two thirds of the instances and it is the only one to solve all of them within the time limit. Its performance is

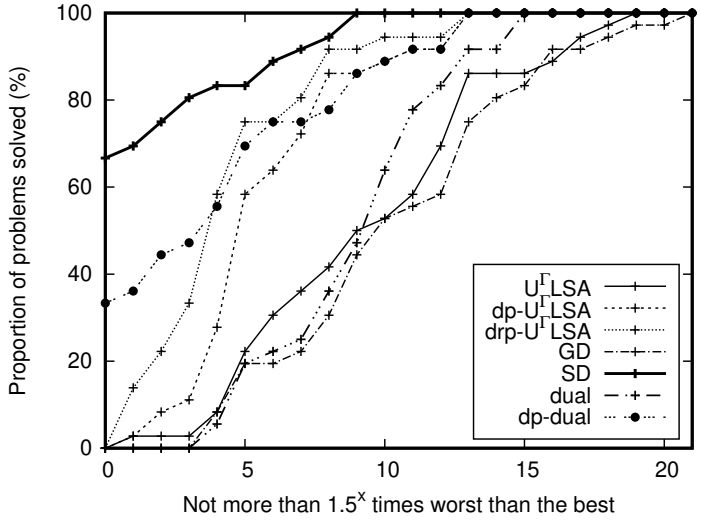

Figure 1: Performance profile.

followed by `dp-dual` and `drp-`$\mathcal{U}^\Gamma$`LSA`, the former being the fastest approach for roughly one third of the instances. While `dp-dual` solve many instances faster than `drp-`$\mathcal{U}^\Gamma$`LSA`, the intersection of the two curves indicates that `drp-`$\mathcal{U}^\Gamma$`LSA` suffer less on the hardest instances than `dp-dual`.

Networks *b*. Table 6 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 7 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 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 6, 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, 7.41

α	Γ	$\mathcal{U}^\Gamma$LSA	dp-$\mathcal{U}^\Gamma$LSA	drp-$\mathcal{U}^\Gamma$LSA	GD	SD	dual	dp-dual
0.25	6	0.219	0.066	0.107	0.116	0.114	0.477	0.449
	3	0.186	0.077	0.098	0.138	0.192	0.887	0.534
	2	0.168	0.068	0.090	0.132	0.244	0.979	0.727
	1	0.155	0.057	0.070	0.078	0.283	0.958	0.380
AVG		0.182	0.067	0.091	0.116	0.208	0.825	0.522
0.5	6	0.257	0.078	0.120	0.139	0.391	1.219	1.263
	3	0.214	0.066	0.095	0.109	0.737	1.182	0.452
	2	0.202	0.059	0.086	0.090	0.967	0.951	0.497
	1	0.181	0.056	0.072	0.099	0.186	0.720	0.482
AVG		0.213	0.065	0.093	0.109	0.570	1.018	0.674
0.75	6	0.298	0.076	0.118	0.251	0.632	0.574	0.197
	3	0.249	0.066	0.094	0.190	1.173	0.447	0.215
	2	0.238	0.066	0.086	0.115	1.493	0.397	0.165
	1	0.219	0.060	0.079	0.103	1.845	0.353	0.148
AVG		0.251	0.067	0.094	0.165	1.286	0.443	0.181
AVG	6	0.258	0.073	0.115	0.169	0.379	0.757	0.636
	3	0.216	0.070	0.095	0.146	0.701	0.839	0.400
	2	0.203	0.064	0.087	0.112	0.902	0.776	0.463
	1	0.185	0.058	0.074	0.093	0.771	0.677	0.337
AVG		0.215	0.066	0.093	0.130	0.688	0.762	0.459

Table 6: Average computational results for each value of Γ and α on networks *b*.

times slower than drp-$\mathcal{U}^\Gamma$LSA. The lower the value of Γ, the higher the speed-up. Indeed, the proposed approach is 3.29, 7.34, 10.33, and 10.45 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 2.28, 6.13, and 13.62 times faster than SD for α equal to 0.25, 0.50, and 0.75, respectively. The comparison with dual and dp-dual is similar to the one with SD, dp-$\mathcal{U}^\Gamma$LSA and drp-$\mathcal{U}^\Gamma$LSA being significantly faster than the algorithms based on dualizations.

The results collected in Table 6 highlight the better behaviour of our approaches than that of GD. Indeed, dp-$\mathcal{U}^\Gamma$LSA and drp-$\mathcal{U}^\Gamma$LSA are, on average, 1.96 and 1.40 times faster than GD.

6.4 Benefit of robustness

On the one hand, the paths returned by the robust algorithms are more expensive than the deterministic paths. Moreover, the cost increases with the value of Γ since the latter characterizes the volume of the uncertainty set. On the other hand, the robust paths are more protected against deviations of the resource consumption than the deterministic paths, and the degree of protection increases with the value of Γ. A natural way to visualize this trade-off considers the resource consumptions as random variables and computes the probability that the paths returned by the robust algorithms do not satisfy the resource

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

constraint.

We suppose in what follows that each resource consumption r_a follows a random variables that takes values $\overline{r}_a$, $\overline{r}_a + \hat{r}_a$, $\overline{r}_a - \hat{r}_a$ with probabilities q, $(1-q)/2$, and $(1-q)/2$, respectively. For simplicity, we assume that these random variables are independent. We would like to compute the probability that the path p returned by each model is unfeasible, that is

$$P\left(\sum_{a\in p} r_a > W\right). \tag{24}$$

Since computing (24) exactly can be cumbersome for paths having 50 edges or more, we rely on Monte Carlo sampling. We generate randomly 10^6 m-uples r^i following the above probabilities, and compute the proportion of scenarios for which the capacity is not satisfied, that is

$$10^{-6} \times \left|\left\{ i \in \{1,\dots,10^6\} : \sum_{a\in p} r_a^i > W \right\}\right|. \tag{25}$$

Let us denote by $\mathbf{P}(\gamma)$ (resp. $\kappa(\gamma)$) the value of (25) (resp. the cost) computed for the optimal solution p obtained for the robust model with $\Gamma = \gamma$. In particular, $\mathbf{P}(0)$ (resp. $\kappa(0)$) is the probability (resp. cost) obtained for the deterministic solution. Next we analyze the value of $\mathbf{P}$ for the different robust models on the instance $G1$ from [11] that is generated for $\Gamma = 6$ (recall from Section 6.2 that the generation of the robust instances depends on Γ). We also compare $\mathbf{P}$ with the theoretical upper bound for the failure probability provided by [6], denoted $P(\gamma)$ hereafter.

In Figure 2, we report $\mathbf{P}(\gamma)$, $P(\gamma)$, and $\kappa(\gamma)$ for γ ranging from 0 to 10 and $q = 0.5$. Comparing $\mathbf{P}$ and P, we see that the empirical bound is much better than the theoretical one, since the latter is valid for any independent and symmetrically distributed random variables. We see on the figure that the failure probability $\mathbf{P}$ can be as low as 0.01 by setting $\gamma = 10$. However, that robust

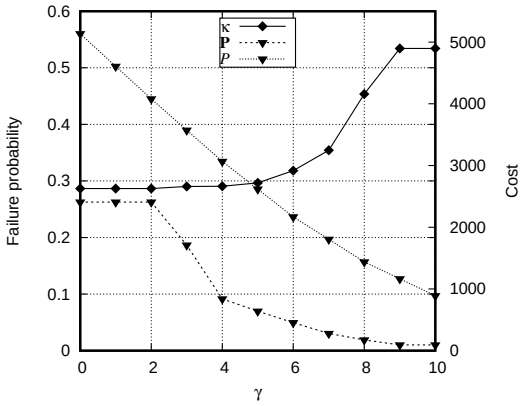

Figure 2: $\mathbf{P}(\gamma)$, $P(\gamma)$ and $\kappa(\gamma)$ on the instance $G1$ created for $\Gamma = 6$.

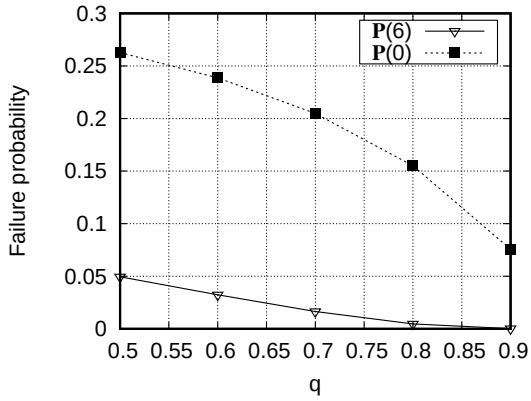

Figure 3: $\mathbf{P}(0)$ and $\mathbf{P}(6)$ for different values of q on the instance $G1$ created for $\Gamma = 6$.

model leads to a solution almost twice as expensive as the deterministic one (4897 vs 2627). This being said, the deterministic solution is highly unreliable since it fails with probability 0.26. As it is always the case when dealing with uncertainty, the problem involves multiple and conflicting objectives and the best solution will depend on the specific application and the risk averseness of the decision maker. If the decision maker is satisfied with a failure probability of 0.05, then he/she might choose the solution obtained for $\Gamma = 6$, which costs only 11% more than the deterministic one (2917 vs 2627).

Let us now focus on the choice of q, which should in practice be estimated from expert knowledge or historical data. For instance, we compare in Figure 3 the failure probability of the paths obtained for $\gamma = 6$ and $\gamma = 0$ (the deterministic model). The figure illustrates that the failure probability of the robust solution falls down to less than 0.001 when $q = 0.9$ while the deterministic solution still has a failure probability of more than 0.075.

7 Conclusions

In this paper, we address a variant of $RCSPP$ where the resources consumptions are uncertain parameters. We consider the budgeted uncertainty set introduced by Bertsimas and Sim [5]. Being weakly $\mathcal{NP}$-hard, the problem can be solved in pseudo-polynomial time using a label-setting algorithm [30]. We study here the effect of incorporating a preprocessing phase in 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 [11] for $RCSPP$. We have generated several robust instances by considering different level of risk aversion of the decision maker.

The computational results show a good behaviour of the preprocessing phase. Indeed, the computed lower and upper bounds can 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, when Γ is not too large (not greater than 6) the label-setting algorithm outperforms the algorithm implementing the well-known iterative algorithm proposed by Bertsimas and Sim in [5] and previously used in [23, 26] as well as the classical dualized integer programming reformulation and the preprocessing and label-setting procedures described in [22]. For larger values of Γ, however, the iterative algorithm is the fastest approach. This difference follows naturally from the fact that the complexity of the label-setting algorithm is exponential in Γ, while the one of the iterative algorithm is only linear in Γ. Finally, we illustrate the benefit of robustness by considering one of the instances from [11]. Our results show that the robust models provide solutions that are more reliable than the deterministic for a reasonable cost increase. They also illustrate empirically how $\Gamma = 6$ can lead to reliable solutions for problems defined on graphs having thousands

of nodes.

Appendix: Detailed numerical results

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

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 for $\Gamma \in \{6, 3, 2, 1\}$. Tables 8 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).

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).

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 8). In Table 9 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

Γ	α	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap
6	0.25	1.94	1.30	23.37	9.12	7.11%	10.00%	34.48%	37.24%	41.59%	47.24%	9.33%
	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%
3	0.25	1.16	1.69	13.84	8.19	13.79%	18.89%	27.27%	30.32%	41.05%	49.21%	2.80%
	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%
2	0.25	0.93	1.87	11.11	8.08	15.84%	22.29%	25.82%	27.79%	41.66%	50.08%	2.30%
	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%
1	0.25	0.69	2.05	7.70	7.72	19.79%	27.24%	22.43%	23.37%	42.22%	50.61%	1.82%
	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 8: Average computational results of the preprocessing phase varying the parameters Γ and α for networks G with $\Gamma \in \{6, 3, 2, 1\}$.

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

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 10 shows the behaviour of the preprocessing procedure for each value of Γ varying γ (see the end of Section 6.2).

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 10: 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 G for $\Gamma \in \{20, 15, 10\}$. Table 11 reports the average computational results over the network $G1$ and $G2$ which are solved by drp-$\mathcal{U}^{\Gamma}$LSA for each value of Γ and α, whereas Table 12 shows the computational results averaged over all networks G for $\alpha \in \{0.50, 0.75\}$.

Γ	α	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap
20	0.25	0.33	0.04	2.19	0.59	3.90%	0.21%	76.54%	3.45%	80.44%	3.65%	1.08%
	0.50	0.29	0.05	2.34	0.65	1.50%	0.14%	76.54%	3.45%	78.04%	3.59%	4.04%
	0.75	0.08	0.01	0.97	0.66	0.18%	0.22%	89.16%	3.27%	89.34%	3.49%	0.15%
AVG		0.23	0.03	1.83	0.63	1.86%	0.19%	80.75%	3.39%	82.61%	3.58%	1.76%
15	0.25	1.05	0.18	6.21	0.44	6.32%	2.38%	68.20%	3.00%	74.52%	5.38%	1.06%
	0.50	0.27	0.04	1.94	0.58	3.02%	0.05%	76.54%	3.45%	79.56%	3.50%	4.04%
	0.75	0.24	0.04	1.83	0.63	0.60%	0.11%	87.64%	4.32%	88.24%	4.43%	0.26%
AVG		0.52	0.09	3.33	0.55	3.31%	0.85%	77.46%	3.59%	80.77%	4.44%	1.79%
10	0.25	0.77	0.30	3.09	0.41	11.24%	3.89%	44.34%	4.70%	55.58%	8.59%	4.86%
	0.50	0.21	0.05	1.59	0.73	4.06%	0.62%	70.26%	4.94%	74.32%	5.56%	3.81%
	0.75	0.12	0.03	0.78	0.78	0.24%	0.42%	88.16%	5.74%	88.40%	6.15%	0.29%
AVG		0.36	0.13	1.82	0.64	5.18%	1.64%	67.59%	5.13%	72.77%	6.77%	2.99%

Table 11: Average computational results of the preprocessing phase varying the parameters Γ and α for networks $G1$ and $G2$ with $\Gamma \in \{20, 15, 10\}$.

The solution times collected in Table 11 do not reveal a trend for the computational overhead when varying Γ. Indeed, the preprocessing requires 2.73, 4.48, and 2.95 seconds, on average, for Γ equal to 20, 15, and 10, respectively. The average results varying α are more interesting. In this case, the higher α, the lower the computational effort. Indeed, the average execution time for solving the networks $G1$ and $G2$ with $\alpha = 0.75$ is 1.41, and 2.53 times faster than solving the same networks with α equal to 0.50 and 0.25, respectively. In addition, the higher α, the higher the percentage of both nodes and arcs removed from the networks. The average %n is 70.18%, 77.31%, and 88.66% for α equal to 0.25, 0.50, and 0.75, whereas the %a is 5.87%, 4.21%, and 4.69%. The better performance for high value of α is mainly due to the network reduction based on cost (see column %nξ and %aξ).

Γ	α	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap
	0.50	7.66	0.36	114.29	20.80	1.49%	0.18%	71.41%	2.96%	72.90%	3.13%	4.09%
	0.75	1.41	0.03	35.54	13.96	0.11%	0.11%	89.44%	2.54%	89.55%	2.66%	0.66%
AVG		4.54	0.19	74.91	17.38	0.80%	0.14%	80.43%	2.75%	81.22%	2.89%	2.37%
	0.50	3.79	0.09	81.51	24.77	1.73%	0.03%	75.35%	2.80%	77.08%	2.83%	3.71%
	0.75	1.11	0.05	27.56	16.05	0.31%	0.08%	89.97%	3.15%	90.28%	3.22%	0.13%
AVG		2.45	0.07	54.54	20.41	1.02%	0.06%	82.66%	2.97%	83.68%	3.03%	1.92%
	0.50	2.75	0.09	49.04	13.23	2.48%	0.37%	69.25%	3.50%	71.73%	3.87%	3.82%
	0.75	0.85	0.04	21.39	16.32	0.13%	0.22%	89.91%	3.90%	90.04%	4.12%	0.21%
AVG		1.80	0.07	35.22	14.78	1.31%	0.29%	79.58%	3.70%	80.89%	3.99%	2.01%

Table 12: Average computational results of the preprocessing phase varying the parameters Γ and $\alpha \in \{0.50, 0.75\}$ for networks G with $\Gamma \in \{20, 15, 10\}$.

Table 12 gives more insight on the preprocessing behaviour for the considered instances. We observe an increase of the execution time when the value of Γ

increases. The preprocessing for $\Gamma = 10$ is, on average, 1.49 and 1.87 times faster than for Γ equal to 15 and 20, respectively. The values of %n and %a remain almost the same for each value of Γ. The preprocessing for $\alpha = 0.75$ is 2.37 times faster than for $\alpha = 0.50$ and the average reduction of both nodes and arcs is higher for $\alpha = 0.75$ than for $\alpha = 0.50$. This behaviour can be justified by the gap. Indeed, for the instances with α equal to 0.50 the gap is 3.87%, while it decreases to 0.33% for α equal to 0.75.

Table 13 shows the average results for each combination of Γ and γ varying α. We display the number of networks solved under column slv.

α	Γ	γ	slv	tw	trw	tξ	trξ	%nw	%aw	%nξ	%aξ	%n	%a	gap
0.25	20	20	2	0.33	0.04	2.19	0.59	3.90%	0.21%	76.54%	3.45%	80.44%	3.65%	1.08%
		15	2	0.20	0.03	1.95	0.57	0.44%	0.01%	76.54%	3.45%	76.98%	3.45%	1.08%
		10	2	0.16	0.02	1.55	0.51	0.10%	0.01%	76.54%	3.45%	76.64%	3.46%	1.08%
		6	2	0.09	0.01	1.06	0.53	0.00%	0.00%	76.54%	3.45%	76.54%	3.45%	31.83%
		3	2	0.06	0.00	1.15	0.55	0.00%	0.00%	76.54%	3.45%	76.54%	3.45%	31.83%
		2	2	0.05	0.00	0.97	0.51	0.00%	0.00%	76.54%	3.45%	76.54%	3.45%	31.83%
		1	2	0.04	0.01	1.05	0.52	0.00%	0.00%	76.54%	3.45%	76.54%	3.45%	31.83%
	15	15	2	0.91	0.16	6.15	0.48	6.32%	2.38%	68.20%	3.00%	74.52%	5.38%	1.06%
		10	2	0.56	0.07	4.45	0.51	0.00%	0.02%	69.24%	4.86%	69.24%	4.88%	2.08%
		6	2	0.29	0.02	2.87	0.49	0.00%	0.00%	70.26%	5.07%	70.26%	5.07%	1.49%
		3	2	0.14	0.02	2.01	0.51	0.00%	0.00%	70.26%	5.07%	70.26%	5.07%	1.49%
		2	2	0.12	0.02	1.77	0.55	0.00%	0.00%	70.26%	5.07%	70.26%	5.07%	1.49%
		1	2	0.09	0.00	1.38	0.00	0.00%	0.00%	0.96%	0.13%	0.96%	0.13%	94.42%
	10	10	2	0.66	0.27	3.09	0.38	11.24%	3.89%	44.34%	4.70%	55.58%	8.59%	4.86%
		6	2	0.30	0.03	1.55	0.48	0.14%	0.04%	32.82%	0.74%	32.96%	0.78%	37.60%
		3	2	0.15	0.01	1.47	0.00	0.00%	0.00%	0.96%	0.13%	0.96%	0.13%	87.73%
		2	2	0.13	0.01	1.10	0.01	0.00%	0.00%	0.96%	0.13%	0.96%	0.13%	87.73%
		1	2	0.08	0.01	1.05	0.02	0.00%	0.00%	0.96%	0.13%	0.96%	0.13%	87.73%
0.50	20	20	4	7.66	0.36	114.29	20.80	1.49%	0.18%	71.41%	2.96%	72.90%	3.13%	4.09%
		15	3	1.64	0.08	16.92	2.21	0.21%	0.02%	74.33%	2.80%	74.55%	2.82%	4.98%
		10	3	1.03	0.05	12.45	2.08	0.00%	0.00%	77.77%	2.88%	77.77%	2.88%	3.47%
		6	3	0.64	0.04	8.17	2.17	0.00%	0.00%	80.19%	3.31%	80.19%	3.31%	1.85%
		3	2	0.08	0.01	1.13	0.52	0.00%	0.00%	79.24%	3.99%	79.24%	3.99%	2.47%
		2	2	0.05	0.02	1.06	0.59	0.00%	0.00%	81.58%	4.47%	81.58%	4.47%	1.10%
		1	2	0.04	0.00	0.95	0.52	0.00%	0.00%	81.58%	4.47%	81.58%	4.47%	1.10%
	15	15	4	3.57	0.08	68.84	13.94	1.73%	0.03%	75.35%	2.80%	77.08%	2.83%	3.71%
		10	4	2.27	0.07	45.36	12.92	0.15%	0.01%	75.39%	2.80%	75.54%	2.82%	1.67%
		6	3	0.20	0.02	2.70	2.51	0.01%	0.00%	77.87%	2.78%	77.88%	2.78%	22.48%
		3	3	0.17	0.02	2.72	2.12	0.00%	0.00%	80.19%	3.31%	80.19%	3.31%	21.51%
		2	3	0.14	0.02	2.65	2.15	0.00%	0.00%	79.67%	3.14%	79.67%	3.14%	21.85%
		1	3	0.11	0.01	2.20	2.13	0.00%	0.00%	79.67%	3.14%	79.67%	3.14%	21.85%
	10	10	4	2.55	0.10	45.32	12.63	2.48%	0.37%	69.25%	3.50%	71.73%	3.87%	3.82%
		6	4	1.60	0.07	37.59	16.44	0.29%	0.01%	71.93%	3.87%	72.23%	3.88%	3.11%
		3	3	0.24	0.03	2.52	2.26	0.11%	0.00%	69.74%	3.87%	69.84%	3.87%	5.20%
		2	3	0.19	0.03	2.02	2.26	0.07%	0.00%	69.74%	3.87%	69.80%	3.87%	5.20%
		1	3	0.13	0.01	1.95	2.07	0.03%	0.01%	70.38%	4.21%	70.40%	4.22%	4.79%
0.75	20	20	4	1.41	0.03	35.54	13.96	0.11%	0.11%	89.44%	2.54%	89.55%	2.66%	0.66%
		15	4	1.11	0.03	35.72	17.23	0.00%	0.00%	89.07%	2.54%	89.07%	2.54%	0.83%
		10	4	0.81	0.02	29.77	16.96	0.00%	0.00%	89.07%	2.54%	89.07%	2.54%	0.83%
		6	4	0.56	0.02	27.53	15.98	0.00%	0.00%	89.07%	2.54%	89.07%	2.54%	0.83%
		3	4	0.47	0.01	24.13	17.77	0.00%	0.00%	89.07%	2.54%	89.07%	2.54%	0.83%
		2	4	0.47	0.02	23.99	15.80	0.00%	0.00%	89.07%	2.54%	89.07%	2.54%	0.83%
		1	4	0.44	0.02	23.06	16.49	0.00%	0.00%	89.07%	2.54%	89.07%	2.54%	0.83%
	15	15	4	1.14	0.03	27.52	16.61	0.31%	0.08%	89.97%	3.15%	90.28%	3.22%	0.13%
		10	4	0.85	0.02	24.15	17.23	0.04%	0.01%	90.15%	3.15%	90.19%	3.15%	0.17%
		6	4	0.61	0.02	20.63	18.41	0.00%	0.00%	89.04%	2.80%	89.04%	2.80%	0.62%
		3	4	0.50	0.02	19.67	17.21	0.00%	0.00%	84.89%	2.74%	84.89%	2.74%	2.75%
		2	4	0.45	0.02	17.16	15.89	0.00%	0.00%	84.50%	2.62%	84.50%	2.62%	3.01%
		1	4	0.42	0.01	16.15	15.59	0.00%	0.00%	84.50%	2.62%	84.50%	2.62%	3.01%
	10	10	4	0.79	0.02	20.87	17.08	0.13%	0.22%	89.91%	3.90%	90.04%	4.12%	0.21%
		6	4	0.59	0.02	20.41	16.41	0.00%	0.00%	90.13%	3.90%	90.13%	3.90%	0.89%
		3	4	0.46	0.01	16.45	16.39	0.00%	0.00%	86.24%	3.23%	86.24%	3.23%	2.34%
		2	4	0.50	0.04	17.78	16.18	0.00%	0.00%	86.24%	3.23%	86.24%	3.23%	2.34%
		1	4	0.41	0.01	15.64	13.51	0.00%	0.00%	86.24%	3.23%	86.24%	3.23%	2.34%

Table 13: Average computational results of the preprocessing phase varying Γ for each value of γ for networks G with $\Gamma \in \{20, 15, 10\}$.

Table 13 highlights the strong effect of γ. Indeed, the lower γ, the lower

the benefits of preprocessing. This is an expected trend, since the lower γ, the lower the quality of the bounds. The reduction of both nodes and arcs reaches zero for several value of γ. It follows that a lower number of networks are solved for lower value of γ. This behaviour is clearly observed for the instances with $\alpha = 0.50$.

Networks b**.** Table 14 shows the value of preprocessing on networks b. We report the number of instances solved to optimality by the preprocessing under column #OPT. Each row of the table report average results over 12 instances. 51% of the instances are solved to optimality. In particular, the lower the value of Γ, the higher the number of instances the preprocessing solves to 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 14: 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 α. However, for higher values of α, 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 α, but a higher number of feasible paths is present for higher values of α.

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 α, a greater number 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 α as shown in 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 α and, consequently for high values of W. Indeed, given a multiplier λ, the cost of the Lagrangean problem is decreased by the constant λW.

In Table 15, we show the performance of the preprocessing on instances with a given value of Γ considering lower bounds computed for $\gamma \leq \Gamma$.

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

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.

Gap closing phase evaluation

Tables 16–22 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 for $\Gamma \in \{6, 3, 2, 1\}$. Table 16 shows average results on instances derived from networks 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 16: Average computational results of the gap closing phase varying Γ for networks G.

The average results, varying the parameter α, are reported in Table 17. Considering the instances with α equal to 0.75 and 0.50, the labeling algorithm is 1549.00 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 17: Average computational results of the gap closing phase varying α for networks G.

Networks G for $\Gamma \in \{20, 15, 10\}$.

Table 18 reports average results varying Γ. As observed for the instances with $\Gamma \in \{6, 3, 2, 1\}$, the higher Γ, the higher the execution time. The time required by the preprocessing and the label-setting procedures for the instances with $\Gamma = 10$ is 1.19 and 1.93 times lower than the computational effort for the instances with Γ equal to 15 and 20, respectively. This trend is mainly due to the computational overhead of the preprocessing. Indeed, it represents the 82% of the execution time required by drp-$\mathcal{U}^\Gamma$LSA.

On the one hand, the fathoming rule that considers the resource consumption is more effective for lower value of Γ (see column #LW). On the other hand, the number of labels discarded by using bounds on cost remains almost unchanged for the three different values of Γ. However, the fathoming by cost is more effective than the fathoming by resource.

Γ	prep	LSA	#L	#LW	#LUB
20	58.66	10.16	48008.08	2674.42	28922.58
15	31.62	10.91	54179.75	3021.50	28089.25
10	30.16	5.57	43312.42	6206.00	20733.25
AVG	40.15	8.88	48500.08	3967.31	25915.03

Table 18: Average computational results of the gap closing phase varying Γ for networks G.

Table 19 reports the average results over the networks $G1$ and $G2$ by varying α. The performance of drp-$\mathcal{U}^\Gamma$LSA is strongly affected by α. In particular, we observe an evident reduction of the execution time of the label-setting algorithm for an increase of α (see column LSA). This behaviour is justified by the number of generated labels. Indeed, the label-setting algorithm explores 28.67 and 16.03 times as many labels for α equal to 0.25 and 0.50 as the number of labels explored by the algorithm for $\alpha = 0.75$.

The same trend is observed considering all the solved networks G for α equal to 0.50 and 0.75. The related average results are reported in Table 20.

In particular, the execution time for solving the instances with $\alpha = 0.75$ by drp-$\mathcal{U}^{\Gamma}$LSA is 2.63 times lower than the execution time required by drp-$\mathcal{U}^{\Gamma}$LSA for solving the instances with $\alpha = 0.50$.

α	prep	LSA	#L	#LW	#LUB
0.25	2.29	1.73	26145.00	6495.00	8959.83
0.50	1.36	0.63	14620.00	1822.83	22123.33
0.75	1.46	0.01	911.83	63.50	463.50
AVG	1.70	0.79	13892.28	2793.78	10515.56

Table 19: Average computational results of the gap closing phase varying α for networks $G1$ and $G2$ with $\Gamma \in \{20, 15, 10\}$.

α	prep	LSA	#L	#LW	#LUB
0.50	84.22	19.42	76768.92	4125.00	59909.25
0.75	33.93	5.49	42586.33	1281.92	8876.00
AVG	59.08	12.45	59677.63	2703.46	34392.63

Table 20: Average computational results of the gap closing phase varying $\alpha \in \{0.50, 0.75\}$ for networks G with $\Gamma \in \{20, 15, 10\}$.

Network b. The labeling algorithm is very fast in solving the instances derived from network b. Table 21 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 #LW.

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

Table 22 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 22: Average computational results of the gap closing phase varying α for networks *b*.

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] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25:1–13, 1999.

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

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

[7] 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.

[8] 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.

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

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

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

[12] A. Elimam and D. Kohler. Two engineering applications of a constrained shortest-path model. *European Journal of Operational Research*, 103(3):426–438, 1997.

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

[14] C. E. Gounaris, P. P. Repoussis, C. D. Tarantilis, W. Wiesemann, and C. A. Floudas. An adaptive memory programming framework for the robust capacitated vehicle routing problem. *Transportation Science*, 50(4):1239–1260, 2016.

[15] C. E. Gounaris, W. Wiesemann, and C. A. Floudas. The robust capacitated vehicle routing problem under demand uncertainty. *Operations Research*, 61(3):677–693, 2013.

[16] J. Halpern and I. Priess. Shortest path with time constraints on movement and parking. *Networks*, 4(3):241–253, 1974.

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

[18] C. Lee, K. Lee, K. Park, and S. Park. Technical note - branch-and-price-and-cut approach to the robust network design problem without flow bifurcations. *Operations Research*, 60(3):604–610, 2012.

[19] C. Lee, K. Lee, and S. Park. Robust vehicle routing problem with deadlines and travel time/demand uncertainty. *Journal of the Operational Research Society*, 63(9):1294–1306, 2012.

[20] T. Lee and C. Kwon. A short note on the robust combinatorial optimization problems with cardinality constrained uncertainty. *4OR*, 12(4):373–378, Dec 2014.

[21] L. Lozano and A. Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40:278–284, 2013.

[22] D. Lu. *Robust optimization for airline scheduling and vehicle routing*. PhD thesis, University of Waterloo, 2014. Available at https://uwspace.uwaterloo.ca/handle/10012/8964.

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

[24] D. Lu and F. Gzara. Models and algorithms for the robust resource constrained shortest path problem, 2018. Available at Optimization Online.

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

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

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

[28] F. Ordónez. Robust vehicle routing. *TUTORIALS in Operations Research*, pages 153–178, 2010.

[29] A. Pessoa, M. Poss, R. Sadykov, and F. Vanderbeck. Branch-and-cut-and-price for the robust capacitated vehicle routing problem with knapsack uncertainty, 2018. Available at Optimization Online.

[30] 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.

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

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

[33] M. Poss. Robust combinatorial optimization with knapsack uncertainty. *Discrete Optimization*, 27:88–102, 2018.

[34] U. Ritzinger, J. Puchinger, and R. F. Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[35] 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.

[36] M. C. Santos, A. Agra, M. Poss, and D. Nace. A dynamic programming approach for a class of robust optimization problems. *SIAM Journal on Optimization*, 3:1799–1823, 2016.

[37] A. Sedeno-Noda and S. Alonso-Rodriguez. An enhanced k-sp algorithm with pruning strategies to solve the constrained shortest path problem. *Applied Mathematics and Computation*, 265:602–618, 2015.

[38] I. Sungur, F. O. nez, and M. Dessouky. A robust optimization approach for the capacitated vehicle routing. *IIE Transactions*, 40(5):509–523, 2008.

[39] L. Wang, L. Yang, and Z. Gao. The constrained shortest path problem with stochastic correlated link travel times. *European Journal of Operational Research*, 255(1):43–57, 2016.

[40] G. Xue. Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees. In *Performance, Computing, and Communications Conference, 2000. IPCCC'00. Conference Proceeding of the IEEE International*, pages 271–277. IEEE, 2000.

[41] M. Zabarankin, S. Uryasev, and P. Pardalos. Optimal risk path algorithms. *Applied Optimization*, 66:273–296, 2002.