Faster Algorithms for Min-max-min Robustness for Combinatorial Problems with Budgeted Uncertainty

André Chassein, Marc Goerigk, Jannis Kurtz, Michael Poss

Faster Algorithms for Min-max-min Robustness for Combinatorial Problems with Budgeted Uncertainty

André Chassein[1], Marc Goerigk[2], Jannis Kurtz[3], and Michael Poss*[4]

[1]Data Analytics Center of Excellence, Deutsche Post DHL Group, Bonn, Germany
[2]Network and Data Science Management, University of Siegen, Germany
[3]Department of Mathematics, RWTH Aachen University, Germany
[4]LIRMM, University of Montpellier, CNRS, France

Abstract

We consider robust combinatorial optimization problems where the decision maker can react to a scenario by choosing from a finite set of k solutions. This approach is appropriate for decision problems under uncertainty where the implementation of decisions requires preparing the ground. We focus on the case that the set of possible scenarios is described through a budgeted uncertainty set and provide three algorithms for the problem. The first algorithm solves heuristically the dualized problem, a non-convex mixed-integer non-linear program (MINLP), via an alternating optimization approach. The second algorithm solves the MINLP exactly for $k = 2$ through a dedicated spatial branch-and-bound algorithm. The third approach enumerates k-tuples, relying on strong bounds to avoid a complete enumeration. We test our methods on shortest path instances that were used in the previous literature and on randomly generated knapsack instances, and find that our methods considerably outperform previous approaches. Many instances that were previously not solved within hours can now be solved within few minutes, often even faster.

Keywords: combinatorial optimization; robust optimization; k-adaptability; budgeted uncertainty; branch-and-bound algorithms

1 Introduction

Real-world problems are uncertain, and optimization approaches need tools to reflect this uncertainty. One such approach is robust optimization, which dates back to the seminal work of Soyster (1973). Since the breakthroughs arising about twenty years ago (Ben-Tal and Nemirovski, 1998, 1999; Kouvelis and Yu, 2013; El Ghaoui et al., 1998), robust optimization has become a key framework to address the uncertainty that arises in optimization problems.

*Corresponding author. Email: michael.poss@lirmm.fr

The rationale behind robust optimization is to characterize the uncertainty over unknown parameters through a set which contains all relevant scenarios and to measure the worst-case over this set. One of the main reasons for the success of robust optimization is its tractability. For instance, linear robust optimization problems are essentially as easy as their deterministic counterparts for many types of convex uncertainty sets, see Ben-Tal and Nemirovski (1998), contrasting with the well-known difficulty of stochastic optimization approaches. For general overviews on the field, we refer to Aissi et al. (2009); Bertsimas et al. (2011); Goerigk and Schöbel (2016); Gabrel et al. (2014).

The picture is more complex when it comes to robust combinatorial optimization problems. Let $[n] = \{1, \dots, n\}$ denote a set of indices and $\mathcal{X} \subseteq \{0,1\}^n$ be the feasibility set of a combinatorial optimization problem. Given a bounded uncertainty set $U \subseteq \mathbb{R}^n$, the classical robust counterpart of the problem $\min_{x\in\mathcal{X}} \sum_{i\in[n]} c_i x_i$ is

$$\min_{x\in\mathcal{X}} \max_{c\in\mathcal{U}} \sum_{i\in[n]} c_i x_i. \tag{M^2}$$

It is well known (e.g. Aissi et al. (2009); Kouvelis and Yu (2013)) that a general uncertainty set $\mathcal{U}$ leads to a robust problem that is, more often than not, harder than the deterministic problem. Robust combinatorial optimization witnessed a breakthrough with the introduction of budgeted uncertainty by Bertsimas and Sim (2003) (also known as Γ-uncertainty), which keeps the tractability of the deterministic counterpart for a large class of combinatorial optimization problems. Specifically, Bertsimas and Sim (2003) considered uncertain cost functions characterized by the vector $\hat{c} \in \mathbb{R}^n$ of nominal costs and the vector $d \in \mathbb{R}^n_+$ of deviations. Then, given a budget of uncertainty $\Gamma > 0$, they addressed uncertainty sets of the form

$$\mathcal{U}^\Gamma = \left\{ c \in \mathbb{R}^n : c_i = \hat{c}_i + d_i z_i,\ z \in [0,1]^n,\ \sum_{i\in[n]} z_i \le \Gamma \right\}.$$

Bertsimas and Sim (2003) showed how the optimal solution of problem (M^2) can be obtained by solving $n+1$ deterministic counterparts of the problem. Several subsequent papers have reduced this number of deterministic problems (Álvarez-Miranda et al., 2013; Lee et al., 2012a), down to $\lceil \frac{n-\Gamma}{2} \rceil + 1$ in Lee and Kwon (2014), and extended the result to more general uncertainty polytopes (Poss, 2018). We refer to Buchheim and Kurtz (2018b); Kasperski and Zieliński (2016) for recent surveys on robust combinatorial optimization.

Our focus in this paper is the alternative robust model introduced by Buchheim and Kurtz (2017, 2018a) which is based on the idea of k-adaptability first introduced by Bertsimas and Caramanis (2010) for general robust two-stage problems. Later this idea was studied for robust two-stage problems with binary recourse in Hanasusanto et al. (2015) and for the same problems with mixed-integer recourse in Subramanyam et al. (2017). In the latter publications both cases, uncertainty only affecting the objective function and uncertainty affecting the constraint coefficients, are studied. In contrast to this, the approach of Buchheim and Kurtz (2017, 2018a) is limited to the case of objective uncertainty and binary decision variables. Furthermore the authors do not consider robust two-stage problems, containing first-stage and second-stage decisions, but apply the idea of k-adaptability to classical combinatorial problems. The main idea of the approach is that the decision maker *prepares* k solutions from $\mathcal{X}$ before knowing the scenario c. Then, upon full knowledge of c, the decision maker can choose the cheapest of the k solutions that had been prepared. Using the robust paradigm the aim is to find k solutions which perform well in the worst case over all scenarios while in the objective function for each scenario the best of the k solutions is considered. This idea results in the problem

$$\min_{x^{(1)},\ldots,x^{(k)}\in\mathcal{X}} \max_{c\in\mathcal{U}} \min_{j\in[k]} \sum_{i\in[n]} c_i x_i^{(j)}. \tag{M^3}$$

The approach modeled by (M^3) is typically useful in applications where some groundwork has to be made ahead of knowing the data. An example taken from Hanasusanto et al. (2015) is related to disaster management wherein one must be able to transport relief supplies or evacuating citizens in case an uncertain disaster arises (see e.g. Chang et al. (2007); Liberatore et al. (2013)). Here the storage locations of the supplies and possible evacuation paths have to be determined in advance. The optimized set of emergency plans, $x^{(1)},\ldots,x^{(k)}\in\mathcal{X}$ for a small number k, ought to be planned and trained for well before the disaster happens.

Similar applications arise in the context of transportation problems in logistics (e.g. the Hub-Location Problem; see Alumur et al. (2012)). Here a company may have to make decisions in advance (e.g. reserving a fleet of trucks or parts of a railing system owned by the government, constructing facilities or hubs etc.) to provide a working supply-chain in the future. Since the future demands of the customers are uncertain, a flexible transportation system is very useful. Using the min-max-min approach a small number of transportation-plans can be calculated in advance to decide which long-term decision have to be made and to prepare all employees. Another example, taken from Eufinger et al. (2018); Subramanyam et al. (2017), considers a parcel service delivering to the same customers every day, i.e. $\mathcal{X}$ is the set of feasible solutions of the vehicle routing problem. At the beginning of each day, the company determines a route taking into account the current traffic situation. In this case again, the drivers need time to be trained for the set of possible routes, to avoid, for instance being stuck in narrow streets with large vehicles. Hence, the set of candidate routes should be small and known ahead of the departures of the drivers.

When the number of solutions is large ($k \geq n+1$), Buchheim and Kurtz (2017) essentially show for general convex uncertainty sets that (M^3) is not harder than its deterministic counterpart. Unfortunately, in applications that require preparing the ground, it is not practical to have too many alternatives, limiting the interest of the approach from Buchheim and Kurtz (2017), which requires $k \geq n+1$. Alternative solutions to (M^3) have also been proposed in a much more general context where there are also decisions that must be taken before the uncertainty is revealed, falling into the framework of two-stage robust optimization, see Hanasusanto et al. (2015) and Subramanyam et al. (2017). The former provides a MILP reformulation involving the linearization of products between binary and real variables, while the latter studies an ad-hoc branch-and-bound algorithm that branches over the assignments of solutions to scenarios. Unfortunately, these two approaches are able to prove optimality only for the smaller instances studied therein. Considering the case of budgeted uncertainty and $k = 2$, Chassein (2017) proved that Problem (M^3) can be solved in polynomial time for the matroid maximization problem, the selection problem and the unconstrained problem while it is strongly NP-hard for the shortest path problem. Despite the theoretical efficiency for several combinatorial problems the procedure derived in Chassein (2017) is not efficient for practical purposes. Furthermore general algorithms applicable even for the NP-hard cases are desired.

The purpose of this work is to overcome these limitations in the case of small k (typically 2 or 3), proposing efficient exact (and heuristic) algorithms for the resulting optimization problems. Our algorithms are tailored for the budgeted uncertainty set $\mathcal{U}^{\Gamma}$ for two main reasons. First, budgeted uncertainty has been successfully used in numerous applications, including transport and logistics (Agra et al., 2018; Lee et al., 2012b), energy production (Bertsimas et al., 2013), telecommunications network design (Koster et al., 2013; Lee et al., 2012a), portfolio selection (Kawas and Thiele, 2017), among many others. Second, the specific structure of the set can be leveraged to provide efficient algorithms. The contributions of this paper can be summarized as follows:

- We propose a local-search heuristic based on the dualized non-linear reformulation, valid for any value of k.

- We solve the non-linear reformulation exactly through a spatial branch-and-bound algorithm, valid for the case $k = 2$. Our algorithm relies on strong lower bounds, tailored for $\mathcal{U}^\Gamma$.

- We provide an enumeration algorithm to solve the problem for small values of k, typically 2 or 3. Leveraging the structure of $\mathcal{U}^\Gamma$, as well as ad-hoc upper and lower bounds, the algorithm is able to enumerate a small subset of the k-tuples to prove optimality.

- Using shortest path instances from the literature and new randomly generated knapsack instances, we show that our methods are able to improve computation times considerably, solving problems to optimality within minutes (often seconds) that were previously unsolved in hours. For $k = 4$, our heuristic provides solutions close to those obtained in the literature in small amounts of time.

The remainder of this paper is structured as follows. The algorithms based on the non-linear reformulation are presented in Section 2 while the general enumerative algorithm is described in Section 3. Computational experiments are discussed in Section 4, before concluding the paper in Section 5.

Notation. For any integer n, we denote the set $\{1, \dots, n\}$ as $[n]$. Further, the k-tuple $(x^{(1)}, \dots, x^{(k)})$ is shortened to $\mathbf{x}$, and $\mathcal{X}^k$ denotes the Cartesian product $\times_{i=1}^k \mathcal{X}$.

2 Non-Linear Algorithms

2.1 Problem Reformulation

Our first two algorithms address problem (M^3) as a non-convex MINLP. Introducing the optimization variable z to express the inner minimization problem of (M^3) leads to the following min-max problem

$$\begin{aligned}
&\min_{\mathbf{x}\in\mathcal{X}^k} \max_{c\in\mathcal{U}^\Gamma} \max_{z} \left\{ z : z \le \sum_{i\in[n]} c_i x_i^{(j)}, \forall j \in [k] \right\} \\
&= \min_{\mathbf{x}\in\mathcal{X}^k} \max_{c,z} \left\{ z : z \le \sum_{i\in[n]} c_i x_i^{(j)}, \forall j \in [k], c \in \mathcal{U}^\Gamma \right\}.
\end{aligned} \tag{1}$$

Dualizing the inner maximization problem, which is a linear optimization problem, we obtain the following non-linear compact formulation for (M^3):

$$\begin{aligned}
\min_{\mathbf{x},\theta,\gamma,\alpha} \quad & \sum_{j\in[k]}\sum_{i\in[n]} \hat{c}_i x_i^{(j)}\alpha_j + \Gamma\theta + \sum_{i\in[n]}\gamma_i \\
\text{s.t.} \quad & \theta + \gamma_i \geq \sum_{j\in[k]} d_i x_i^{(j)}\alpha_j && \forall i \in [n] \\
& \sum_{j\in[k]} \alpha_j = 1 \\
& \theta \geq 0 \\
& \gamma_i \geq 0 && \forall i \in [n] \\
& \alpha_j \geq 0 && \forall j \in [k] \\
& \mathbf{x} \in \mathcal{X}^k.
\end{aligned} \tag{NL}$$

Notice that the above formulation can be linearized by replacing the product $\alpha_j x_i^{(j)}$ by a new variable z_i^j, which is restricted by the constraints $z_i^j \geq 0$ and $z_i^j \geq \alpha_j + x_i^{(j)} - 1$. This leads to a compact mixed-integer programming formulation (MIP) (Hanasusanto et al., 2015).

Chassein (2017) showed that it suffices to enumerate a finite set of $O(n^{2k-1})$ many values for α and θ, to solve Problem (NL) exactly. Note that if α and θ is fixed the non-linearity vanishes and the problem reduces to an MIP with a certain structure. The following theorem, proved in Chassein (2017), summarizes this result in detail.

Theorem 1. *Chassein (2017) Problem* (M^3) *with budgeted uncertainty can be solved by solving at most $O(n^{2k-1})$ subproblems of the form*

$$\min_{\mathbf{x}\in\mathcal{X}^k} \sum_{i\in[n]} f_i(x_i^{(1)},\ldots,x_i^{(k)}) \tag{P_{sub}}$$

where

$$f_i(x_i^{(1)},\ldots,x_i^{(k)}) := \sum_{j\in[k]} \alpha_j \hat{c}_i x_i^{(j)} + \max\left(0, \sum_{j\in[k]} \alpha_j d_i x_i^{(j)} - \theta\right)$$

for some fixed values $\alpha \in \mathbb{R}_+^k$ and $\theta \in \mathbb{R}_+$.

Chassein (2017) proves that Problem (P_{sub}) can be solved in polynomial time for the matroid maximization problem, the selection problem, the unconstrained problem and the shortest path problem on series-parallel graphs. Despite these positive results the author shows that Problem (M^3) is strongly NP-hard for the shortest path problem in general.

2.2 Local Search Heuristic

The nonlinear part of model (NL) is due to the product between $x^{(j)}$ and α. A simple idea to avoid the nonlinearity is to search for local instead of global minima by considering only restricted search directions. Methods of this type are also known as block-coordinate descent algorithms, see, e.g. Wright (2015). Instead of minimizing $\mathbf{x}, \alpha, \gamma$, and θ simultaneously, either we solve (NL) only over the variables $\mathbf{x}, \gamma, \theta$ for fixed values of α, or we solve (NL) over the variables α, γ, θ and keep $\mathbf{x}$ fixed. The first optimization problem is called x-step, the second α-step. To solve an x-step, we solve the following MIP

$$\begin{aligned}
\min_{\mathbf{x},\gamma,\theta} \quad & \sum_{j\in[k]}\sum_{i\in[n]} \alpha^j \hat{c}_i x_i^{(j)} + \sum_{i\in[n]} \gamma_i + \Gamma\theta && && (x\text{-step})\\
\text{s.t.} \quad & \sum_{j\in[k]} \alpha^j d_i x_i^{(j)} - \theta \le \gamma_i && \forall i \in [n]\\
& \gamma_i \ge 0 && \forall i \in [n]\\
& \theta \ge 0\\
& \mathbf{x} \in \mathcal{X}^k && \forall j \in [k]
\end{aligned}$$

To solve an α-step, we solve the following LP

$$\begin{aligned}
\min_{\alpha,\gamma,\theta} \quad & \sum_{j\in[k]}\sum_{i\in[n]} \alpha^j \hat{c}_i x_i^{(j)} + \sum_{i\in[n]} \gamma_i + \Gamma\theta && && (\alpha\text{-step})\\
\text{s.t.} \quad & \sum_{j\in[k]} \alpha_j = 1\\
& \sum_{j\in[k]} \alpha^j d_i x_i^{(j)} - \theta \le \gamma_i && \forall i \in [n]\\
& \gamma_i \ge 0 && \forall i \in [n]\\
& \alpha_j \ge 0 && \forall j \in [k]\\
& \theta \ge 0
\end{aligned}$$

We start the local search with an x-step. As initial values for α we choose $\tilde{\alpha}^j = \frac{2j}{k(k+1)}$ for all $j \in [k]$. Note that $\sum_{j\in[k]} \tilde{\alpha}^j = 1$. Different values for $\tilde{\alpha}$ help to break the symmetry of the model formulation. The optimal solution of the x-step is then used to solve the first α-step. We iterate between x- and α-steps until no further improvement is found. Note that we can use the optimal solution of an x-step to warm start the next x-step. Since the objective value decreases in each step, except for the last step, we will end up in a local minimum after a finite number of steps.

2.3 A Branch-and-Bound Algorithm for $k = 2$

If $k = 2$, Problem (NL) can be rewritten in the following way (see also Chassein (2017)):

$$\begin{aligned} \min_{x,y,\alpha,\gamma,\theta} \ & \sum_{i\in[n]} \hat{c}_i x_i \alpha + \sum_{i\in[n]} \hat{c}_i y_i (1-\alpha) + \sum_{i\in[n]} \gamma_i + \Gamma\theta \\ \text{s.t. } & d_i x_i \alpha + d_i y_i (1-\alpha) - \theta \le \gamma_i && \forall i \in [n] \\ & \gamma_i \ge 0 && \forall i \in [n] \\ & \alpha \in [0, 0.5] \\ & \theta \ge 0 \\ & x, y \in \mathcal{X} \end{aligned} \tag{NL-2}$$

In the following we define the optimal value of Problem (NL-2) for a fixed value of $\alpha \in [0, 0.5]$ by $h(\alpha)$. Hence, our original problem can be solved if we can solve the problem $\min_{\alpha\in[0,0.5]} h(\alpha)$. From Theorem 1, we know that the candidate set $\mathcal{A}$ of optimal values for α is a finite set with size $O(n^3)$. Hence, a possible algorithm for the problem is to evaluate $h(\alpha)$ for each $\alpha \in \mathcal{A}$ and choose the best solution. However, solving $O(n^3)$ of these MIPs can be too time consuming. Using the structure of h we can find the global minimum without evaluating $h(\alpha)$ for each $\alpha \in \mathcal{A}$.

The idea of the algorithm is to use a branch-and-bound strategy on the α variable and to divide the interval $[0, 0.5]$ into smaller intervals. For each unexplored interval we calculate lower bounds which are described in detail below. If the list of unexplored intervals is empty, the algorithm has found the optimal solution. There are two reasons which allow to discard an interval. First, if for an interval $\mathcal{I}$ it holds $\mathcal{A} \cap \mathcal{I} = \emptyset$ then it can be discarded since we know that the optimal solution is attained for an $\alpha \in \mathcal{A}$. Second, if the lower bound for the actual interval exceeds the current best solution, the interval can be discarded as well.

If we cannot discard an interval $\mathcal{I} = [\alpha_1, \alpha_2]$ we evaluate $h(\tilde{\alpha})$ for some $\tilde{\alpha} \in \mathcal{I}$. This allows us to split $\mathcal{I}$ into two smaller sub intervals $[\alpha_1, \tilde{\alpha}]$ and $[\tilde{\alpha}, \alpha_2]$. These intervals are then added to the list of unexplored intervals. It is possible that $h(\tilde{\alpha})$ improves the current best solution, which leads to an improved upper bound.

To get a good feasible solution at the start of the algorithm, we use the local search heuristic from Section 2.2 to find a local minimum $h(\alpha^*)$ at α^*. The first two intervals of the list of unexplored intervals are then given as $[0, \alpha^*]$ and $[\alpha^*, 0.5]$.

For the effectiveness of this algorithm the computation of the lower bound is crucial. We argue in the following how to derive a strong lower bound which is still reasonable to compute.

It was shown in Chassein (2017) that $h(\alpha) = \min_{x,y\in\mathcal{X}} g(x,y,\alpha)$ where

$$g(x,y,\alpha) = \alpha \sum_{i\in[n]} \hat{c}_i x_i + (1-\alpha)\sum_{i\in[n]} \hat{c}_i y_i + \left\| \begin{pmatrix} d_1 x_1 \alpha + d_1 y_1 (1-\alpha) \\ \vdots \\ d_n x_n \alpha + d_n y_n (1-\alpha) \end{pmatrix} \right\|^{(\Gamma)}$$

and $||v||^{(\Gamma)}$ is the sum of the Γ largest values of vector v. Note that $g(x,y,\alpha)$ is a piecewise affine-linear function in α where the breakpoints are the values of α for which the order of the components in $||\cdot||^{(\Gamma)}$ changes. If we increase α, clearly each time when the order changes the slope of the next segment must increase. Therefore g is convex in α. Hence, we have that

$$g(x,y,\alpha) \geq g(x,y,\alpha_0) + (\alpha - \alpha_0)\partial g(x,y,\alpha_0)$$

where $\partial g(x,y,\alpha_0)$ is a subdifferential for g. Recall that ∂g is given by

$$\partial g(x,y,\alpha_0) = \sum_{i\in[n]} \hat{c}_i x - \sum_{i\in[n]} \hat{c}_i y_i + \sum_{i\in I: x_i=1, y_i=0} d_i - \sum_{i\in I: x_i=0, y_i=1} d_i$$

where I is the set of the Γ largest indices of $\begin{pmatrix} d_1 x_1 \alpha_0 + d_1 y_1 (1-\alpha_0) \\ \vdots \\ d_n x_n \alpha_0 + d_n y_n (1-\alpha_0) \end{pmatrix}$. The following two estimations are essential to compute the lower bound:

$$\partial g(x,y,\alpha_0) \geq \sum_{i\in[n]} \hat{c}_i x_i - \sum_{i\in[n]} \hat{c}_i y_i - \left\| \begin{pmatrix} d_1 y_1 \\ \vdots \\ d_n y_n \end{pmatrix} \right\|^{(\Gamma)} =: \underline{\partial g}(x,y)$$

and

$$\partial g(x,y,\alpha_0) \leq \sum_{i\in[n]} \hat{c}_i x_i - \sum_{i\in[n]} \hat{c}_i y_i + \left\| \begin{pmatrix} d_1 x_1 \\ \vdots \\ d_n x_n \end{pmatrix} \right\|^{(\Gamma)} =: \overline{\partial g}(x,y).$$

Given an interval $\mathcal{I} = [\alpha_1, \alpha_2]$ for which we want to find a lower bound value $L(\mathcal{I})$ with $L(\mathcal{I}) \leq \min_{\alpha\in[\alpha_1,\alpha_2]} h(\alpha)$ the idea is to solve the following two minimization problems

$$\min_{x,y} g(x,y,\alpha_1) + (\alpha_2 - \alpha_1)\underline{\partial g}(x,y) \tag{2}$$

and

$$\min_{x,y} g(x,y,\alpha_2) + (\alpha_1 - \alpha_2)\overline{\partial g}(x,y). \tag{3}$$

Lemma 2. *Let (x_1^*, y_1^*) be an optimal solution of problem (2) and (x_2^*, y_2^*) an optimal solution of problem (3). For all $\alpha \in [\alpha_1, \alpha_2]$ it holds that*

$$h(\alpha) \geq h(\alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x_1^*, y_1^*)$$

and

$$h(\alpha) \geq h(\alpha_2) + (\alpha - \alpha_2)\overline{\partial g}(x_2^*, y_2^*).$$

Proof. Let $\alpha \in [\alpha_1, \alpha_2]$ be fixed. Let (x^*, y^*) be a solution which defines $h(\alpha)$. For the sake of contradiction, assume that $h(\alpha) = g(x^*, y^*, \alpha) < h(\alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x_1^*, y_1^*)$. First, observe that

$$\begin{aligned} h(\alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x_1^*, y_1^*) &> g(x^*, y^*, \alpha) \\ &\geq g(x^*, y^*, \alpha_1) + (\alpha - \alpha_1)\partial g(x^*, y^*) \\ &\geq g(x^*, y^*, \alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x^*, y^*) \\ &\geq h(\alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x^*, y^*) \end{aligned}$$

From this, it follows that $\underline{\partial g}(x_1^*, y_1^*) > \underline{\partial g}(x^*, y^*)$. Further, we have that

$$\begin{aligned} g(x_1^*, y_1^*, \alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x_1^*, y_1^*) &\geq h(\alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x_1^*, y_1^*) \\ &> g(x^*, y^*, \alpha) \\ &\geq g(x^*, y^*, \alpha_1) + (\alpha - \alpha_1)\partial g(x^*, y^*) \\ &\geq g(x^*, y^*, \alpha_1) + (\alpha - \alpha_1)\underline{\partial g}(x^*, y^*). \end{aligned}$$

Since $\underline{\partial g}(x_1^*, y_1^*) > \underline{\partial g}(x^*, y^*)$, we can add on the left hand side of this inequality chain $(\alpha_2 - \alpha)\underline{\partial g}(x_1^*, y_1^*)$ and on the right hand side $(\alpha_2 - \alpha)\underline{\partial g}(x^*, y^*)$ and obtain

$$g(x_1^*, y_1^*, \alpha_1) + (\alpha_2 - \alpha_1)\underline{\partial g}(x_1^*, y_1^*) > g(x^*, y^*, \alpha_1) + (\alpha_2 - \alpha_1)\underline{\partial g}(x^*, y^*)$$

This gives the desired contradiction since (x_1^*, y_1^*) is an optimal solution for problem (2).

The second inequality can be proved analogously. □

Using Lemma 2 we obtain the following result.

Theorem 3. *Let (x_1^*, y_1^*) be an optimal solution of problem* (2) *and (x_2^*, y_2^*) an optimal solution of problem* (3)*. For $\mathcal{I} = [\alpha_1, \alpha_2]$ a lower bound $L(\mathcal{I})$ is given by*

$$L(\mathcal{I}) = h(\alpha_1) + \left(\frac{h(\alpha_2) - h(\alpha_1) + \alpha_1 \underline{\partial g}(x_1^*, y_1^*) + \alpha_2 \overline{\partial g}(x_2^*, y_2^*)}{\underline{\partial g}(x_1^*, y_1^*) + \overline{\partial g}(x_2^*, y_2^*)} - \alpha_1 \right) \underline{\partial g}(x_1^*, y_1^*).$$

Proof. Let $\alpha \in [\alpha_1, \alpha_2]$. We know from Lemma 2 that $h(\alpha) \geq f_1(\alpha)$ and $h(\alpha) \geq f_2(\alpha)$ where f_1 and f_2 are the two linear functions given in Lemma 2 . We conclude, that $h(\alpha) \geq \max\{f_1(\alpha), f_2(\alpha)\}$. Hence, $\min_{\alpha \in [\alpha_1, \alpha_2]} h(\alpha) \geq \min_{\alpha \in [\alpha_1, \alpha_2]} \max\{f_1(\alpha), f_2(\alpha)\}$. Note that the value of the right hand side is given by $f_1(\alpha')$ where $f_1(\alpha') = f_2(\alpha')$. Using the formulas for f_1 and f_2, we obtain that $L(\mathcal{I}) = f_1(\alpha')$. □

In our branch-and-bound procedure we will use value α' from the proof above as a candidate to split the interval $[\alpha_1, \alpha_2]$ into two smaller intervals $[\alpha_1, \alpha']$ and $[\alpha', \alpha_2]$.

To use Theorem 3 in the branch-and-bound algorithm, we need to know the following four values: $h(\alpha_1)$, $h(\alpha_2)$, $\underline{\partial g}(x_1^*, y_1^*)$, and $\overline{\partial g}(x_2^*, y_2^*)$. The first two values are already computed by the algorithm, since we compute $h(\alpha')$ if we split an interval $[\alpha_1, \alpha_2]$ into two smaller intervals $[\alpha_1, \alpha']$ and $[\alpha', \alpha_2]$. Hence we know for each unexplored interval the value of h at the boundaries of this interval (at the start of the algorithm we also compute $h(0)$ and $h(0.5)$). To compute $\underline{\partial g}(x_1^*, y_1^*)$, and $\overline{\partial g}(x_2^*, y_2^*)$, we need to solve problems (2) and (3). Each of these problems can be formulated as an MIP, which is explained in the following.

For fixed x, y the value of $g(x, y, \alpha_1)$ can be represented by Problem (NL-2) fixing $\alpha = \alpha_1$. Next, consider the value of

$$\underline{\partial g}(x, y) = \hat{c}^\top x - \hat{c}^\top y - \left\| \begin{pmatrix} d_1 y_1 \\ \vdots \\ d_n y_n \end{pmatrix} \right\|^{(\Gamma)}.$$

We introduce variables $\beta_i \in [0, 1]$ for $i \in [n]$ which indicate the Γ largest entries of $(d_1 y_1, \ldots, d_n y_n)$. Therefore calculating $\underline{\partial g}(x, y)$ results in the following minimization problems

$$\begin{aligned} \min_{\beta} \ & \hat{c}^\top x - \hat{c}^\top y - \sum_{i \in [n]} d_i \beta_i \\ \text{s.t.} \ & \sum_{i \in [n]} \beta_i \leq \Gamma \\ & 0 \leq \beta_i \leq y_i \quad \forall i \in [n]. \end{aligned} \tag{4}$$

Recall that the objective function of problem (2) is given by $g(x, y, \alpha_1) + (\alpha_2 - \alpha_1)\underline{\partial g}(x, y)$. Therefore substituting Formulations (NL-2) for fixed $\alpha = \alpha_1$ and Formulation (4) in Problem (2) we obtain the equivalent MIP formulation

$$\begin{aligned}
\min_{x,y,\beta,\gamma,\theta} \quad & \sum_{i\in[n]} \hat{c}_i x_i \alpha_2 + \sum_{i\in[n]} \hat{c}_i y_i (1-\alpha_2) + \sum_{i\in[n]} \gamma_i + \Gamma\theta + \sum_{i\in[n]} (\alpha_1 - \alpha_2) d_i \beta_i \\
\text{s.t.} \quad & d_i x_i \alpha_1 + d_i y_i (1-\alpha_1) - \theta \le \gamma_i && \forall i \in [n] \\
& 0 \le \gamma_i && \forall i \in [n] \\
& 0 \le \theta \\
& \sum_{i\in[n]} \beta_i \le \Gamma \\
& 0 \le \beta_i \le y_i && \forall i \in [n] \\
& x, y \in \mathcal{X}.
\end{aligned}$$

Analogously Problem (3) can be reformulated as an MIP. This concludes the discussion on how to compute a lower bound $L(\mathcal{I})$. We summarize the described procedure in Algorithm 1. Note that it is possible to adapt the naïve implementation of this algorithm to make it computationally more effective in practice. For example, whenever the current best solution is improved by evaluating $h(\alpha')$, we can restart the local search heuristic at α' to find a new local minimum.

3 Enumerative Algorithm

Let us consider the set of feasible solutions to the deterministic combinatorial optimization problem as an ordered set, $\mathcal{X} = (x_1, \dots, x_r)$, which we assume to know; we further explain how to compute $\mathcal{X}$ in Section 3.5. Let us reformulate problem (M^3) as

$$\min_{\mathbf{x}\in\mathcal{X}^k} \mathit{cost}(\mathbf{x}), \tag{5}$$

where $\mathit{cost}(\mathbf{x})$ denotes the max-min cost of solution $\mathbf{x}$, that is

$$\mathit{cost}(\mathbf{x}) = \max_{c\in\mathcal{U}} \min_{j\in[k]} c^\top x^{(j)}. \tag{6}$$

Recall that we show in (1) that (6) can be reformulated as a linear program

$$\mathit{cost}(\mathbf{x}) = \max_{c,z} \left\{ z : c \in \mathcal{U}^\Gamma, z \le \sum_{i\in[n]} c_i x_i^{(j)}, \forall j \in [k] \right\}. \tag{7}$$

Algorithm 1: Branch and bound algorithm (BB) with $k = 2$.

```
Compute the candidate set $\mathcal{A}$;
Use the local search heuristic to find a local minimum $h(\alpha^*)$ at $\alpha^*$;
Initialize the list of unexplored intervals $\mathcal{L} = \{[0, \alpha^*], [\alpha^*, 0.5]\}$;
Compute $h(0)$ and $h(0.5)$;
$UB \leftarrow \min(h(0), h(\alpha^*), h(0.5))$;
Solve problems (2) and (3) for $[0, \alpha^*]$;
Solve problems (2) and (3) for $[\alpha^*, 0.5]$;
Use Theorem 3 to compute $L([0, \alpha^*])$;
Use Theorem 3 to compute $L([\alpha^*, 0.5])$;
$LB \leftarrow \min_{\mathcal{I} \in \mathcal{L}} L(\mathcal{I})$;
while $\mathcal{L} \neq \emptyset$ do
    Choose $\mathcal{I}' = \operatorname{argmin}_{\mathcal{I} \in \mathcal{L}} L(\mathcal{I})$, with $\mathcal{I}' = [\alpha_1, \alpha_2]$;
    $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{I}'$;
    Choose $\alpha' \in \mathcal{I}'$ which defines $L(\mathcal{I}')$ (see the proof of Theorem 3);
    Compute $h(\alpha')$;
    Update $UB = \min\{UB, h(\alpha')\}$;
    Solve problems (2) and (3) for $[\alpha_1, \alpha']$;
    Solve problems (2) and (3) for $[\alpha', \alpha_2]$;
    Use Theorem 3 to compute $L([\alpha_1, \alpha'])$;
    Use Theorem 3 to compute $L([\alpha', \alpha_2])$;
    if $L([\alpha_1, \alpha']) < UB$ and $[\alpha_1, \alpha'] \cap \mathcal{A} \neq \emptyset$ then
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{[\alpha_1, \alpha']\}$
    if $L([\alpha', \alpha_2]) < UB$ and $[\alpha', \alpha_2] \cap \mathcal{A} \neq \emptyset$ then
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{[\alpha', \alpha_2]\}$
    $LB \leftarrow \min_{\mathcal{I} \in \mathcal{L}} L(\mathcal{I})$;
Return: $UB$
```

The algorithm described in this section enumerates over all non-symmetric k-tuples $\mathbf{x} \in \mathcal{X}^k$, using upper and lower bounds to prune part of the k-tuples and to avoid computing $cost(\mathbf{x})$ for all k-tuples. We also introduce the concept of *resistance* to enumerate even less elements of $\mathcal{X}^k$. The pseudo-code is provided in Algorithm 2 for the case $k = 2$. Throughout the section, we denote by d^x the vector $(d_i x_i, i \in [n])$, sorted such that $d_1^x \geq d_2^x \geq \cdots \geq d_n^x$.

Algorithm 2: Enumerative algorithm (EA) illustrated for $k = 2$.

```
1  Let $UB$ be the initial upper bound from (8);
2  Compute $\mathcal{X}$;
3  Compute $lb(x)$ for each $x \in \mathcal{X}$ ;
4  repeat
5      Compute $\gamma^q(x)$ for each $x \in \mathcal{X}$;
6      Construct the partition $\mathcal{X} = \bigcup_{\omega \in [q \cdot \Gamma]} \mathcal{X}_\omega^q$;
7      Let $r_\omega = |\mathcal{X}_\omega^q|$ for each $\omega$;
8      foreach $\omega_1$ in $\{q \cdot \Gamma, q \cdot \Gamma - 1, \ldots, 1\}$ do
9          foreach $s_1$ in $\{1, \ldots, r_{\omega_1}\}$ do
10             foreach $\omega_2$ in $\{q \cdot \Gamma, q \cdot \Gamma - 1, \ldots, q \cdot \Gamma + 1 - \omega_1\}$ do
11                 if $\omega_1 = \omega_2$ then $s_2^{first} = s_1 + 1$ ;
12                 else $s_2^{first} = 1$ ;
13                 foreach $s_2$ in $\{s_2^{first}, \ldots, r_{\omega_2}\}$ do
14                     $\mathbf{x} = (x^{(1)}, x^{(2)}) \leftarrow (x_{s_1}, x_{s_2})$;
15                     Compute $LB_1(\mathbf{x}) = \min\left(lb(x^{(1)}), lb(x^{(2)})\right)$ ;
16                     if $LB_1(\mathbf{x}) > UB$ then
17                         continue
18                     else
19                         Compute $LB_2(\mathbf{x})$ using a greedy algorithm ;
20                         if $LB_2(\mathbf{x}) > UB$ then
21                             continue
22                         else
23                             Compute $cost(\mathbf{x})$ by solving (7) ;
24                             if $cost(\mathbf{x}) < UB$ then
25                                 $UB \leftarrow cost(\mathbf{x})$;
26                                 break all for-loops ;
27             $\mathcal{X} \leftarrow \mathcal{X} \setminus \{x_{r_1}\}$;
28 until $UB$ is not updated;
   Return: the $k$-tuple with minimum cost
```

3.1 Upper Bounds

Clearly an optimal solution of the classical robust Problem (M^2) gives an upper bound for Problem (M^3) since in the latter problem choosing the classical robust solution for each of the k solutions we obtain the same objective value for both problems. As starting upper bound UB on the optimal solution cost we choose

$$UB = \min(rob\_opt, heur), \tag{8}$$

where *rob_opt* is the optimal value of the classical robust problem (M^2), obtained using the iterative algorithm from Bertsimas and Sim (2003), and *heur* is the solution obtained by the local search algorithm from Section 2.2. The upper bound UB is improved when a better feasible solution is found in the course of the algorithm.

Observation 4. *We only need to enumerate solutions $x \in \mathcal{X}$ with $\hat{c}^\top x < UB$, since otherwise for every scenario $c \in \mathcal{U}^\Gamma$ we have $c^\top x \geq UB$ and therefore adding x to a solution never improves the current upper bound.*

3.2 Lower Bounds

As computing $cost(\mathbf{x})$ is time-consuming, we avoid computing its value exactly for many k-tuples and calculate instead two lower bounds, denoted by $LB_1(\mathbf{x})$ and $LB_2(\mathbf{x})$, defined below. Every time the cost of a k-tuple must be computed, we first compute $LB_1(\mathbf{x})$, which is done in $\mathcal{O}(k)$. If $LB_1(\mathbf{x}) < UB$, then we compute $LB_2(\mathbf{x})$, requiring $\mathcal{O}(k\Gamma)$ steps. Only if $LB_2(\mathbf{x}) < UB$ we compute $cost(\mathbf{x})$. We notice that these two bounds do not converge to UB, as is the case in many branch-and-bound algorithms. Hence, if the algorithm stops due to the time limit, it only returns a feasible solution to the problem, the remaining optimality gap being meaningless.

Each of the above bounds is derived by considering a particular scenario from $\mathcal{U}^\Gamma$. The first lower bound $LB_1(\mathbf{x})$ considers the scenario that assigns Γ/k deviations per solution. To reduce the computational burden of computing $LB_1(\mathbf{x})$ to a minimum, once the enumeration has started, we compute in a pre-processing step (see line 3 of Algorithm 2) the cost of each solution x, by adding the $\lfloor \frac{\Gamma}{k} \rfloor$ largest deviations and a fraction of the $\lceil \frac{\Gamma}{k} \rceil$-th largest to the nominal costs of x. Formally we define

$$lb(x) = \hat{c}^\top x + \sum_{i=1}^{\lfloor \Gamma/k \rfloor} d_i^x + \left(\frac{\Gamma}{k} - \left\lfloor \frac{\Gamma}{k} \right\rfloor \right) d_{\lceil \Gamma/k \rceil}^x .$$

Once $lb(x)$ is computed for each $x \in \mathcal{X}$, the lower bound can be obtained in $\mathcal{O}(k)$ through

$$LB_1(\mathbf{x}) = \min_{j\in[k]}(lb(x^{(j)})). \tag{9}$$

The second lower bound, denoted $LB_2(\mathbf{x})$ computes a good scenario greedily by taking the current solution $\mathbf{x}$ and affecting the Γ deviations sequentially to the solution having the smallest cost so far, which is the nominal cost plus the deviations already chosen.

While the bounds significantly speed-up the computation of *cost*$(\mathbf{x})$, the cardinality of $\mathcal{X}^k$ is likely to be large. Fortunately, the majority of k-tuples in $\mathcal{X}^k$ can be discarded by using the concept of *resistance* introduced next.

3.3 Resistance

Given an upper bound UB and $q \in \mathbb{N}$, we define the discrete *q-resistance* $\gamma^q(x)$ of any $x \in \mathcal{X}$ as the smallest amount of deviation ω/q ($\omega \in \mathbb{N}$) that need to be affected to d^x such that the cost of x exceeds UB:

$$\gamma^q(x) = \min_{\omega\in\mathbb{N}}\left\{\omega : \hat{c}^\top x + \sum_{i=1}^{\lfloor\omega/q\rfloor} d_i^x + (\omega/q - \lfloor\omega/q\rfloor)d^x_{\lceil\omega/q\rceil} \geq UB\right\}. \tag{10}$$

Notice that if ω/q is integer, the third term of (10) vanishes. What is more, the value of $\gamma^q(x)$ is bounded above by $q\cdot\Gamma$. To see this, suppose there exists a solution $x \in \mathcal{X}$ such that $\gamma^q(x) > q\cdot\Gamma$. Then, the cost of the k-tuple $\mathbf{x} = (x,\dots,x)$ satisfies *cost*$(\mathbf{x}) < UB$. By definition of $\mathbf{x}$, *cost*$(\mathbf{x})$ coincides with the classical robust value of x, denoted by *rob_opt*(x). Hence, $UB \leq$ *rob_opt*$(x) =$ *cost*$(\mathbf{x}) < UB$, which is a contradiction.

We show next that $\gamma^q(x)$ satisfies another crucial property.

Lemma 5. *If* $\mathbf{x} \in \mathcal{X}^k$ *with* $\sum_{j\in[k]}\gamma^q(x^{(j)}) \leq q\cdot\Gamma$, *then* *cost*$(\mathbf{x}) \geq UB$.

Proof. Let π^x be the permutation of $[n]$ used to obtain d^x from the vector $(d_i x_i, i \in [n])$, that is, $d_i^x = d_{\pi^x(i)}$. For each $j \in [k]$, we define the vector $z^{(j)}$ as

$$z_i^{(j)} = \begin{cases} 1 & \text{if } \pi^{x^{(j)}} \leq \lfloor\gamma^q(x^{(j)})/q\rfloor \\ \omega/q - \lfloor\omega/q\rfloor & \text{if } \pi^{x^{(j)}} = \lceil\gamma^q(x^{(j)})/q\rceil \\ 0 & \text{otherwise} \end{cases}.$$

From (10), we see that $\sum_{i\in[n]}(\hat{c}_i + z_i^{(j)}d_i)x^{(j)} \geq UB$. Next, we define $z_i^* = \max_{j\in[k]} z_i^{(j)}$ for each $i \in [n]$, and we have

$$\sum_{i\in[n]}(\hat{c}_i + z_i^* d_i)x^{(j)} \geq UB. \tag{11}$$

Since $\sum_{j\in[k]} \gamma^q(x^{(j)}) \leq q \cdot \Gamma$, we have that $\sum_{i\in[n]} z_i^* \leq \Gamma$ so that

$$(\hat{c}_i + z_i d_i, i \in [n]) \in \mathcal{U}^\Gamma. \tag{12}$$

Statements (11) and (12) imply that $cost(\mathbf{x}) \geq UB$. □

Thanks to the lemma above, we only have to enumerate solutions $\mathbf{x} \in \mathcal{X}^k$ with $\sum_{j\in[k]}\gamma^q(x^{(j)}) > q \cdot \Gamma$. Let us define $\mathcal{X}^q_\omega = \{x \in \mathcal{X} : \gamma^q(x) = \omega\}$ for each $\omega \in \{0, \dots, q \cdot \Gamma\}$. We have that $\mathcal{X} = \cup_{\omega=0}^{q\cdot\Gamma} \mathcal{X}^q_\omega$. The following observation states that we do not need to consider the elements from $\mathcal{X}^q_0$.

Observation 6. *For any $x \in \mathcal{X}^q_0$ it holds $\hat{c}^\top x \geq UB$.*

Following Lemma 5 and Observations 4 and 6, we need to enumerate only the subset of all k-tuples defined by

$$\mathcal{X}^{k,q}_\Gamma = \left\{ \mathbf{x} \in \mathcal{X}^k : x^{(j)} \in \mathcal{X}^q_{\omega_j}, \omega_j \in [q \cdot \Gamma], j \in [k], \sum_{j\in[k]} \omega_j > q \cdot \Gamma \right\}.$$

Notice that the sets $\mathcal{X}^q_\omega$ can be updated every time a better upper bound is found, which explains the presence of **break** in line 26 of Algorithm 2. In particular, the cardinality $|\mathcal{X}^q_0|$ increases with UB. Hence, Observation 6 implies that restarting the enumeration (step 4 of Algorithm 2) when improving UB possibly leads to the removal of many elements of $\mathcal{X}$ at each restart.

3.4 Handling Symmetry

The symmetry among the elements of $\mathcal{X}^k$ can be used to reduce the set of feasible solutions. Specifically, if $\mathbf{x}$ and $\mathbf{x}'$ are made of the same elements of $\mathcal{X}$, but listed in different orders, $cost(\mathbf{x}) = cost(\mathbf{x}')$. Hence, we focus in what follows on the set $\mathbf{X}^k \subset \mathcal{X}^k$, defined as $\mathbf{X}^k = \{(x_{s_1}, \dots, x_{s_k}) \in \mathcal{X}^k : s_j < s_{j+1}, j = 1, \dots, k-1\}$, and we define $\mathbf{X}^{k,q}_\Gamma$ analogously to Section 3.3.

3.5 Computing $\mathcal{X}$

In this section we detail two algorithms to enumerate all elements of $\mathcal{X}' = \{x \in \mathcal{X} : \hat{c}^\top x < UB\}$. Our first approach relies on an ad-hoc branch-and-bound algorithm that enumerates all feasible solutions to the problem

$$\min_{x\in\mathcal{X}, \hat{c}^\top x < UB} \hat{c}^\top x,$$

by branching iteratively on all variables, and collecting all leaves having accumulated n branching constraints. To avoid exploring the full branch-and-bound tree (which would contain 2^n leaves for $\mathcal{X} = \{0,1\}^n$), the algorithm combines UB with the bound provided by a relaxation to prune parts of the tree. Specifically, let $\mathcal{X}^{LP}$ be a formulation for $\mathcal{X}$, that is, a polytope such that $\mathcal{X}^{LP} \cap \{0,1\}^n = \mathcal{X}$, and let us introduce the branching constraints through the disjoint sets $O, Z \subseteq [n]$. At each node of the branch-and-bound tree, the algorithm solves the relaxation $\mathcal{X}^{LP}$ together with the branching constraints accumulated so far

$$\begin{aligned} z^{LP} \;=\; \min \; & \sum_{i \in [n]} \hat{c}_i x_i \\ \text{s.t. } & x \in \mathcal{X}^{LP} \\ & x_i = 0 \qquad \forall i \in Z \\ & x_i = 1 \qquad \forall i \in O \end{aligned} \tag{LP}$$

Nodes of the tree are pruned either because (LP) is infeasible or because $z^{LP} \geq UB$.

The above approach can be improved significantly for problems for which $\mathcal{X}$ can be enumerated through recursive algorithms, such as the knapsack problem, the shortest path problem, the traveling salesman problem, and spanning (Steiner) tree problems, among many others. In that situation, one can embed the constraint $\hat{c}^\top x < UB$ in the recursive algorithms, allowing us to generate all elements of $\mathcal{X}'$ for problems of reasonable dimensions. Let us detail this approach for problem of finding a shortest path from s to t in an undirected graph G with positive costs $\hat{c}$, considering a Depth First Search (DFS). We execute first the Dijkstra algorithm from t to compute the distance between t and each node v in the graph $d(v,t)$. Then, every time a node v is discovered by the DFS along a path P, starting from s, we further consider its successors only if $\sum_{i \in P} \hat{c}_i + d(v,t) < UB$.

We compare the two algorithms in our computational experiments.

4 Computational Results

The aim of this section is two-fold. First, we assess the cost reduction offered by the min-max-min model (M^3) when compared to the classical min-max robust model (M^2). Second, we evaluate in detail the numerical efficiency of the proposed exact and heuristic solution algorithms. Our experiments are carried out on a set of shortest path instances previously used by Hanasusanto et al. (2015) and on randomly generated instances for the min-knapsack problem. In what follows, HKW denotes linearized compact formulation from Section 2.1

(previously used in Hanasusanto et al. (2015)), heur stands for the heuristic algorithm from Section 2.2, BB denotes the branch-and-bound Algorithm 1 from Section 2.3, EA denotes the enumeration Algorithm 2 from Section 3.

All LPs and MIPs involved are solved by CPLEX version 12.6. Algorithms from Section 2 and Section 3 are implemented in C++ and julia, respectively. The local search uses a processor i5-3470 running at 3.2 Ghz while the exact algorithms use a processor Intel X5460 running at 3.16 GHz, respectively. Note that for the experiments in Hanasusanto et al. (2015), Gurobi Optimizer 5.6 was used, but no details are provided on their computer speed. All solution times are reported in seconds.

4.1 Shortest Path Problem (SP)

4.1.1 Instances

We use the shortest path instances presented in Hanasusanto et al. (2015). Each instance is described by three parameters: the number of nodes $|V| \in \{15 + 5i : i \in [7]\}$ of the underlying graph $G = (V, E)$, the number $k \in \{2, 3, 4\}$ of candidate solutions, and the parameter $\Gamma \in \{3, 6\}$ which specifies the size of the uncertainty set. For each parameter combination 100 instances are randomly generated, which results in 4200 instances in total. For the heuristic algorithm presented in Section 2.2, we set the time limit of the x-step to 300 seconds. For all 4200 problem instances this time limit was only met 6 times. For the exact approaches, we set a time limit of 7200 seconds for each experiment, as in Hanasusanto et al. (2015).

4.1.2 Solution Costs

We present in Table 1 the cost reductions obtained by each algorithm compared to the cost provided by the robust model (M^2). Specifically, for each algorithm A, we report

$$\text{cost\_red}(A) = 100 \times \frac{\text{opt}\,(\text{M}^2) - \text{opt}(A)}{\text{opt}\,(\text{M}^2)}. \tag{13}$$

Notice that the three exact algorithms (BB, EA, and HKW) leave some instances unsolved (see the next section for details), explaining why two approaches A and A' for a given value of k may lead to different values for cost_red(A). We see from the table that EA always obtains the highest value, followed closely by the three other approaches. In particular, the table underlines that the feasible solutions calculated by HKW are of good quality, even though

		$k = 2$				$k = 3$			$k = 4$	
Γ	$\|V\|$	EA	BB	heur	HKW	EA	heur	HKW	heur	HKW
3	20	7.6	7.6	7.5	7.6	9.2	9.0	9.2	9.3	9.5
	25	8.6	8.6	8.6	8.6	10.7	10.6	10.7	11.0	11.3
	30	8.7	8.7	8.6	8.7	11.3	11.1	11.3	11.9	12.2
	35	9.0	9.0	8.9	9.0	12.0	12.0	12.0	12.8	13.1
	40	9.0	9.0	9.0	9.0	12.1	12.0	12.0	12.9	13.2
	45	8.6	8.6	8.5	8.6	11.8	11.7	11.8	12.9	13.1
	50	8.6	8.6	8.4	8.6	11.8	11.7	11.7	13.0	13.1
	AVG	8.5	8.5	8.4	8.5	10.8	10.7	10.8	11.2	11.5
6	20	6.7	6.7	6.6	6.7	10.2	10.1	10.2	11.2	11.4
	25	8.2	8.2	8.1	8.2	12.2	12.1	12.2	13.6	13.8
	30	8.6	8.6	8.5	8.6	13.0	12.9	13.0	15.0	15.1
	35	9.2	9.2	9.1	9.2	13.9	13.8	13.9	16.3	16.3
	40	9.5	9.5	9.4	9.5	14.3	14.2	14.2	16.7	16.6
	45	9.8	9.7	9.6	9.7	14.5	14.4	14.4	17.0	16.9
	50	9.8	9.7	9.6	9.7	14.5	14.4	14.4	17.0	16.9
	AVG	8.2	8.2	8.1	8.2	12.3	12.3	12.3	14.0	14.1

Table 1: Cost reduction cost_red(A) for each algorithm A for the SP.

the algorithm cannot prove optimality for most instances and often finishes with optimality gaps greater than 5% (see Tables 3 and 4). The quality of the heuristic is also very good, following closely the results of HKW, even improving over the latter in some cases ($k = 4$, $\Gamma = 6$ and $|V| \geq 40$). Last, the table illustrates the decreasing benefit of increasing the value of k. While the cost reduction is important for $k = 2$, the subsequent improvements are much smaller, in particular for $\Gamma = 3$.

4.1.3 Solution Times

We first present the solution times of the heuristic algorithm, before investigating in detail the results of the three exact algorithms. The average run times of the heuristic are reported in Table 2 for each group of 100 instances. The vast majority of instances are solved within a minute, often in a few seconds.

We next turn to the exact methods and provide a more detailed presentation of the results. We present in Table 3 a comparison of the exact solution times of BB, EA using the DFS strategy described in Section 3.5, and HKW for $k = 2$. Computation times include the means and standard deviations over the 100 instances of each group (unsolved instances count for 7200 seconds). In the last two columns we report the average percental gap between

	$\Gamma = 3$			$\Gamma = 6$		
$\lvert V\rvert$	$k = 2$	$k = 3$	$k = 4$	$k = 2$	$k = 3$	$k = 4$
20	0.1	0.1	0.2	0.2	0.2	0.2
25	0.2	0.3	0.2	0.3	0.6	0.6
30	0.4	0.6	0.5	0.7	1.3	2.3
35	0.7	1.1	1.1	1.6	4.2	7.5
40	1.0	1.3	1.6	3.3	7.8	19.9
45	1.7	2.5	3.5	6.3	18.8	39.7
50	2.1	3.0	4.9	12.1	29.6	73.5

Table 2: Computation times of heur in seconds for the SP.

		Time (seconds)		% Solved			% Gap	
Γ	$\lvert V\rvert$	BB	EA	BB	EA	HKW	BB	HKW
	20	9.9±8.1	0.2±0.8	100	100	100	0.00	0.00
	25	19.5±13.4	0.2±0.7	100	100	99	0.00	3.68
	30	53.4±57.4	0.9±3.0	100	100	69	0.00	5.69
3	35	92.4±103.6	0.9±1.0	100	100	17	0.00	5.70
	40	168.2±176.3	2.0±2.9	100	100	6	0.00	5.96
	45	385.3±429.6	8.0±16.4	100	100	0	0.00	6.48
	50	654.9±700.3	12.9±17.9	100	100	0	0.00	6.75
	20	186.1±832.8	0.2±0.8	99	100	100	8.0e-6	0.00
	25	153.2±358.2	0.6±0.6	100	100	100	0.00	8.00
	30	651.8±930.9	2.8±3.7	100	100	67	0.00	10.72
6	35	1984.2±2075.2	9.0±10.9	92	100	16	7.3e-3	10.76
	40	3484.7±2723.7	26.3±34.0	72	100	5	1.8e-2	11.29
	45	4897.6±2538.6	117.7±312.5	54	100	0	1.2e-1	11.79
	50	6188.8±1951.9	318.6±577.2	30	99	0	2.3e-1	12.31

Table 3: Comparison of HKW, EA, and BB for $k = 2$ for the SP.

upper and lower bound which was reached after the time limit of 7200 seconds. We see that all instances were solved to optimality by EA in a few seconds, and to near optimality by BB. It turns out that the instances of the smaller uncertainty set ($\Gamma = 3$) are easier to solve by our methods. For some instances of the larger uncertainty set the time limit was reached by BB. However, the remaining gap between upper and lower bound reported by BB is reasonably small as reported in Table 3. Notice that algorithm EA returns no optimality gap when it fails to solve the problem to optimality.

Table 3 clearly shows that both EA and BB outperform HKW by orders of magnitude. Further, EA is also much faster than BB. Notice, however, that EA requires large amounts of memory: the only unsolved instance by EA failed because of a memory hit, using a computer with 48 GB of memory. In fact, many large instances require more than 20 GB

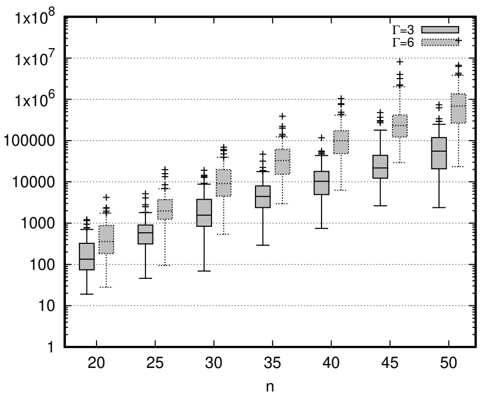
Γ	$\|V\|$	Time EA (seconds)	% Solved		% Gap
			EA	HKW	HWK
3	20	80.7±261.0	100	97	1.60
	25	249.3±1027.0	98	31	1.14
	30	621.7±1539.8	97	6	1.71
	35	741.0±1795.5	95	0	2.23
	40	1232.8±2322.8	90	0	2.59
	45	1871.9±2832.0	82	0	3.14
	50	2210.7±2817.2	80	0	3.44
6	20	49.2±172.6	100	97	4.06
	25	249.4±1025.0	99	38	3.52
	30	593.4±1455.9	97	6	4.54
	35	1133.6±1964.7	96	0	5.58
	40	2466.6±2646.6	82	0	6.19
	45	4529.1±2846.3	53	0	6.92
	50	6021.5±2122.8	29	0	7.55

Table 4: Comparison of HKW and EA for $k = 3$ for the SP.

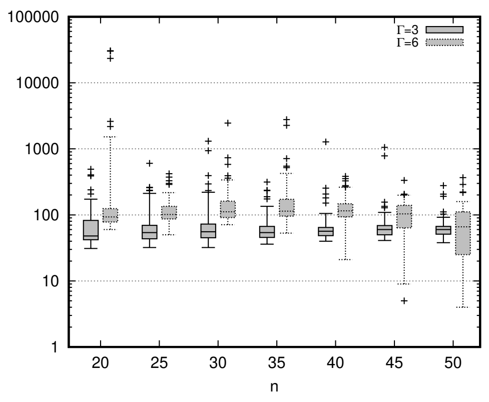
of memory, while BB can handle all instances with a few GBs. The difference in memory consumption is due to the large cardinality of the set $\mathcal{X}$ and the small number of nodes explored by BB. These two aspects are further investigated in Figure 1. Figure 1(a) shows that the number of solutions x that satisfy $\hat{c}^\top x < UB$ increases nearly exponentially with $|V|$, reaching roughly 8×10^8 for $|V| = 50$ and $\Gamma = 6$. In contrast, Figure 1(b) shows that the number of nodes explored by BB does not seem impacted by $|V|$ and revolves around 100 nodes, while Figure 1(c) even shows the root gaps decrease with $|V|$. The three charts of Figure 1 also indicate that $\Gamma = 6$ leads to significantly harder instances than $\Gamma = 3$.

Figure 2 investigates the effect of the *resistance* and the lower bounds, described in Sections 3.3 and 3.2, respectively. Recall that, without using resistance, the number of 2-tuples enumerated should be $\frac{|\mathcal{X}|(|\mathcal{X}|-1)}{2}$. Hence, $|\mathcal{X}| \sim 100$ should lead to 5×10^4 2-tuples, which is far from the results shown on the plain boxes from Figure 2. For instance, consider the case $\Gamma = 6$ and $|V| = 50$. Figure 1(a) shows that for nearly 95% of the instances, $|\mathcal{X}| \geq 10^5$. Yet, Figure 2(b) shows that more than 95% of the instances enumerate at most 10^5 2-tuples. Regarding the interest of the lower bounds, the dotted boxplots from Figure 2 show that the number of 2-tuples for which $cost(x_{s_1}, x_{s_2})$ is actually computed is between half and one order of magnitude less than the number of 2-tuples enumerated.

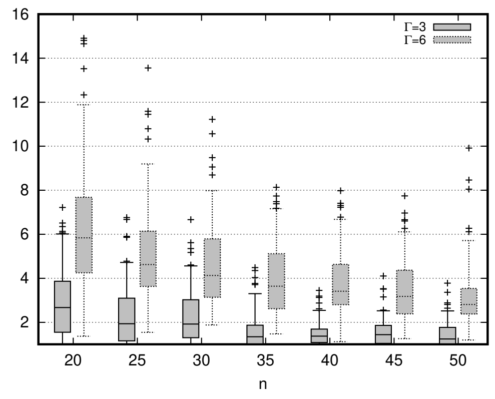
We present in Table 4 a comparison of the exact solution times of EA and HKW for $k = 3$. While EA cannot solve all instances during the time limit, it still outperforms HKW



(a) $|\mathcal{X}|$

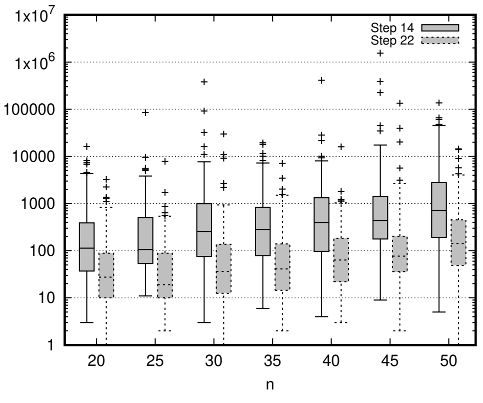


(b) nodes explored by BB

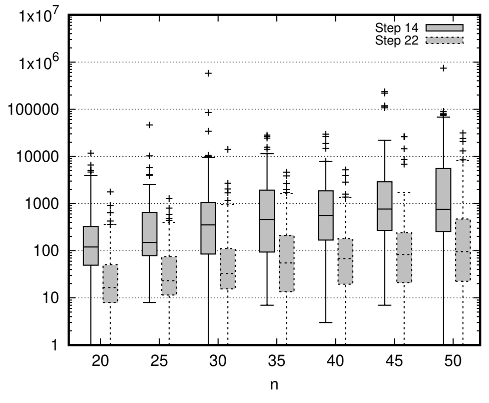


(c) root gap

Figure 1: Size of $\mathcal{X}$ for the SP generated in Step 2 of Algorithm 2, number of nodes explored by BB, in logarithmic scale, and the root gap of BB. The box contains 50% of the data samples, cut by the median. The whiskers extend the box to cover 95% of the samples.



(a) $\Gamma = 3$



(b) $\Gamma = 6$

Figure 2: Number of 2-tuples (x_{s_1}, x_{s_2}) handled by Algorithm 2 at steps 15 and 23, in logarithmic scale for the SP.

	$\Gamma = 3$		$\Gamma = 6$	
$\lvert V\rvert$	Time (seconds)	% Solved	Time (seconds)	% Solved
20	3.1±3.2	100	3.3±3.7	100
25	16.2±15.4	100	16.3±15.9	100
30	100.5±114.1	100	105.1±116.6	100
35	347.1±362.4	100	358.5±372.1	100
40	1148.2±1146.8	99	1174.0±1163.0	99
45	3310.7±2281.9	85	3348.0±2288.9	85
50	5374.4±2283.9	45	5402.5±2263.2	45

Table 5: Results of EA generating $\mathcal{X}$ using a vanilla branch-and-bound algorithm for $k = 2$ and $\Gamma = 3$ for the SP.

significantly, solving 642 ($\Gamma = 3$) and 556 ($\Gamma = 6$) instances to optimality (out of 700), instead of 134 and 141 for HKW.

The results presented for EA so far have relied on the DFS strategy to iterate through the set $\mathcal{X}$. To understand whether the alternative LP-based branch-and-bound algorithm is a realistic way to iterate through $\mathcal{X}$, we have also coded a vanilla version of that branch-and-bound algorithm for the shortest path problem. The latter is coded in julia, using package JuMP, and does not implement advanced warm-starts when processing a new node. The results presented in Table 5 indicate that this strategy is orders of magnitude slower than the DFS. Yet, it is able to solve nearly all instances having less than 50 nodes during the time limit. Nearly 100% of the time is spent in the generation of $\mathcal{X}$.

4.2 Knapsack Problem (KP)

4.2.1 Instances

Since the paper has studied minimization optimization problems so far, we consider next a minimization version of the knapsack problem defined as

$$\min \left\{ \sum_{i\in[n]} c_i x_i : \sum_{i\in[n]} w_i x_i \geq W,\ x \in \{0,1\}^n \right\}.$$

For each dimension n the costs c_i and the weights w_i were drawn from a uniform distribution on $\{1, \dots, 100\}$. The knapsack capacity W was set to 35% of the sum of all weights. For each knapsack instance we generate a budgeted uncertainty set with mean vector $\hat{c} = c$ and a random deviation vector d where each d_i is drawn uniformly in $\{1, \dots, c_i\}$. Each instance is described by three parameters: the number of items $|n| \in \{50i : i \in [4]\}$, the number $k \in \{2, 3, 4\}$ of candidate solutions, and the parameter $\Gamma \in \{3, 6\}$ which specifies

		$k = 2$				$k = 3$			$k = 4$	
Γ	n	EA	BB	heur	HKW	EA	heur	HKW	heur	HKW
3	50	2.0	2.0	1.9	1.9	2.5	2.4	2.1	2.5	2.3
	100	2.0	2.0	2.0	1.6	2.0	2.4	1.4	2.5	1.2
	150	1.2	1.2	1.2	0.3	1.2	1.5	0.0	1.6	0.0
	200	0.5	0.5	0.5	0.0	0.5	0.7	0.0	0.8	0.0
	AVG	1.4	1.4	1.4	1.0	1.5	1.7	0.9	1.9	0.9
6	50	1.5	1.5	1.4	1.5	2.2	2.1	1.6	2.4	1.4
	100	2.4	2.5	2.4	2.0	2.4	3.2	1.4	3.4	1.0
	150	2.0	2.0	2.0	0.4	2.0	2.4	0.7	2.6	0.1
	200	1.1	1.1	1.1	0.0	1.1	1.3	0.0	1.4	0.0
	AVG	1.8	1.8	1.7	1.0	1.9	2.3	0.9	2.5	0.6

Table 6: Cost reduction cost_red(A) for each algorithm A and value of k for the KP.

the size of the uncertainty set. For each parameter combination 10 instances are randomly generated, which results in 240 instances in total. We set a time limit of 3600 seconds for each experiment. The approach HKW mentioned in the following reports on our implementation of the linearized formulation described in Section 2.

4.2.2 Solution Costs

We present in Table 6 the application of formula (13) to the knapsack instances. As for the shortest path, the three exact algorithms (BB, EA, and HKW) leave some instances unsolved, see the next section for details. The table shows that the cost reductions for the knapsack problem are much smaller than those obtained for the shortest path problem, and these reductions decrease with the size of the instances. These smaller cost reductions are possibly due to the smaller ratios of Γ over the number of non-zero binary variables in the optimal solution of the knapsack in comparison to the shortest path. Furthermore, problem structure may play a role, such as the possibilty to pack items with good cost-to-weight ratio in every knapsack solution. We also see that the heuristic solutions are usually better than the best solutions returned by HKW, which is not surprising given the high gaps returned by the latter (see Table 8).

4.2.3 Solution Times

We first present the solution times of the heuristic algorithm. The average run times of the heuristic are reported in Table 7 for each group of 100 instances. As before, the vast majority of instances are solved within a minute, often in a few seconds.

	$\Gamma = 3$			$\Gamma = 6$		
n	$k = 2$	$k = 3$	$k = 4$	$k = 2$	$k = 3$	$k = 4$
50	0.0	0.1	0.1	0.0	0.1	0.1
100	0.1	0.3	1.0	0.2	0.6	3.9
150	0.2	1.5	8.9	0.3	2.7	56.8
200	0.4	1.3	15.2	0.5	3.9	34.8

Table 7: Computation times of heur in seconds for the KP.

		Time (seconds)		% Solved			% Gap	
Γ	n	BB	EA	BB	EA	HKW	BB	HKW
3	50	10.9±6.7	73.9±73.6	100	100	0	0	14.0
	100	32.3±29.9	3600	100	0	0	0	69.9
	150	411.7±1063.5	3600	90	0	0	2.8e-2	90.0
	200	636.6±1037.2	3600	90	0	0	1.2	96.0
6	50	69.6±124.8	259.8±328.5	100	100	0	0	15.2
	100	351.9±811.5	3600	100	0	0	0	71.1
	150	540.9±550.1	3600	100	0	0	0	90.2
	200	1905.9±1244.4	3600	70	0	0	1.8	96.0

Table 8: Comparison of HKW, EA, and BB for $k = 2$ for the KP.

Table 8 compares HKW, BB, and EA on the knapsack instances for $k = 2$. The following observations arise from results presented in the table. First, HKW is not able to solve *any* of the instances, often resulting in large optimality gaps. The inefficiency of HKW is explained by its extremely weak continuous relaxation, the optimal solution of which is 0 for all instances. Second, EA is only able to solve the smallest instances. For 100 items or more, it cannot finish the first step of the algorithm (enumerating the solutions). As it turns out, the knapsack problem is particularly difficult for EA because that problem lacks strong constraints on the structure of the feasible solutions, so that there exists plenty of similar solutions that cannot be removed before the algorithm starts. Third, as for the shortest path, BB can solve to optimality most of the instances, ending with small optimality gaps for the unsolved ones. The efficiency of BB is mainly due to its tight root gap, as further illustrated in Figure 3.

For $k = 3$, EA is able to solve 7 out of 10 instances for $\Gamma = 3$, and 4 out of 10 instances for $\Gamma = 6$, while the HKW solves none of them, ending with average gaps of 64% and 66%, respectively. Larger instances cannot be solved by EA and HKW ends up with gaps above 90%.

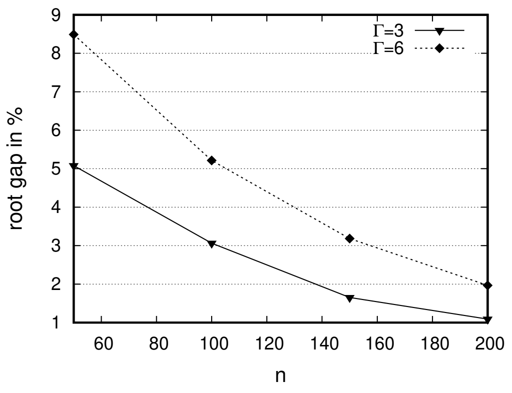


Figure 3: Root gaps for the KP.

5 Conclusions

Min-max-min combinatorial optimization problems form a class of notoriously difficult optimization problems. In this manuscript, we have provided a first step towards their efficient solution, focusing on the case of budgeted uncertainty. In this work we derived three fast algorithms to solve these problems exactly and heuristically, two of them based on a mixed-integer non-linear reformulation while the third is a discrete enumeration scheme involving ad-hoc dominance rules. The proposed exact algorithms have overcome the difficulties encountered by the previous literature (Hanasusanto et al., 2015; Subramanyam et al., 2017) for $k = 2$, by solving problems in a couple of minutes (often seconds) that were unsolvable in one or two hours using the previous approaches. We also found encouraging results for $k = 3$, solving many unsolved instances to optimality in a short amount of time. While our exact approaches can hardly handle larger values of k, our local search heuristic based on the non-linear reformulation performs well, providing near-optimal solutions quickly.

In the future, we intend to develop solution algorithms able to solve the problem exactly for larger values of k and more general uncertainty sets. One idea is to use strong integer programming tools based on extended formulations. Preliminary results seem to indicate that the linear programming relaxation of these formulations are strong, hopefully leading to efficient branch-and-bound algorithms.

6 Acknowledgements

This research benefited from the support of the FMJH Program PGMO and from the support to this program from EDF-THALES-ORANGE- CRITEO. The authors are also grateful to

the referees for their helpful suggestions.

References

Agostinho Agra, Marielle Christiansen, Lars Magnus Hvattum, and Filipe Rodrigues. Robust optimization for a maritime inventory routing problem. *Transportation Science*, 52(3): 509–525, 2018.

Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438, 2009.

Sibel A Alumur, Stefan Nickel, and Francisco Saldanha-da Gama. Hub location under uncertainty. *Transportation Research Part B: Methodological*, 46(4):529–543, 2012.

Eduardo Álvarez-Miranda, Ivana Ljubic, and Paolo Toth. A note on the Bertsimas & Sim algorithm for robust combinatorial optimization problems. *4OR*, 11(4):349–360, 2013.

Aharon Ben-Tal and Arkadi Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998.

Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13, 1999.

Dimitris Bertsimas and Constantine Caramanis. Finite adaptability in multistage linear optimization. *IEEE Transactions on Automatic Control*, 55(12):2751–2766, 2010.

Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Math. Program.*, 98(1-3):49–71, 2003.

Dimitris Bertsimas, David B Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM review*, 53(3):464–501, 2011.

Dimitris Bertsimas, Eugene Litvinov, Xu Andy Sun, Jinye Zhao, and Tongxin Zheng. Adaptive robust optimization for the security constrained unit commitment problem. *IEEE Transactions on Power Systems*, 28(1):52–63, 2013.

Christoph Buchheim and Jannis Kurtz. Min–max–min robust combinatorial optimization. *Mathematical Programming*, 163(1):1–23, 2017.

Christoph Buchheim and Jannis Kurtz. Complexity of min–max–min robustness for combinatorial optimization under discrete uncertainty. *Discrete Optimization*, 28:1–15, 2018a.

Christoph Buchheim and Jannis Kurtz. Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO Journal on Computational Optimization*, 6(3):211–238, 2018b.

Mei-Shiang Chang, Ya-Ling Tseng, and Jing-Wen Chen. A scenario planning approach for the flood emergency logistics preparation problem under uncertainty. *Transportation Research Part E: Logistics and Transportation Review*, 43(6):737–754, 2007.

André Chassein. Having a plan B for robust optimization. Technical report, Technische Universität Kaiserslautern, 2017.

Laurent El Ghaoui, Francois Oustry, and Hervé Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52, 1998.

Lars Eufinger, Jannis Kurtz, Christoph Buchheim, and Uwe Clausen. A robust approach to the capacitated vehicle routing problem with uncertain costs. Technical report, Optimization Online, 2018.

Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471–483, 2014.

Marc Goerigk and Anita Schöbel. Algorithm engineering in robust optimization. In *Algorithm engineering*, pages 245–279. Springer, 2016.

Grani A Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891, 2015.

Adam Kasperski and Paweł Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. In *Robustness analysis in decision aiding, optimization, and analytics*, pages 113–143. Springer, 2016.

Ban Kawas and Aurélie Thiele. Log-robust portfolio management with parameter ambiguity. *Comput. Manag. Science*, 14(2):229–256, 2017.

Arie M. C. A. Koster, Manuel Kutschka, and Christian Raack. Robust network design: Formulations, valid inequalities, and computations. *Networks*, 61(2):128–149, 2013.

Panos Kouvelis and Gang Yu. *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media, 2013.

Chungmok Lee, Kyungsik Lee, Kyungchul Park, and Sungsoo Park. Technical note - branch-and-price-and-cut approach to the robust network design problem without flow bifurcations. *Operations Research*, 60(3):604–610, 2012a.

Chungmok Lee, Kyungsik Lee, and Sungsoo Park. Robust vehicle routing problem with deadlines and travel time/demand uncertainty. *Journal of the Operational Research Society*, 63(9):1294–1306, 2012b.

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

F Liberatore, C Pizarro, C Simón de Blas, MT Ortuño, and B Vitoriano. Uncertainty in humanitarian logistics for disaster management. a review. In *Decision aid models for disaster management and emergencies*, pages 45–74. Springer, 2013.

Michael Poss. Robust combinatorial optimization with knapsack uncertainty. *Discrete Optimization*, 27:88–102, 2018.

Allen L Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

Anirudh Subramanyam, Chrysanthos E Gounaris, and Wolfram Wiesemann. K-adaptability in two-stage mixed-integer robust optimization. *arXiv preprint arXiv:1706.07097*, 2017.

Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.