Min-Max-Min Robustness for Combinatorial Problems with Discrete Budgeted Uncertainty

Marc Goerigk, Jannis Kurtz, Michael Poss

Min-Max-Min Robustness for Combinatorial Problems with Discrete Budgeted Uncertainty

Marc Goerigk

Network and Data Science Management, University of Siegen, Germany

Jannis Kurtz

Department of Mathematics, RWTH Aachen University, Germany

Michael Poss

LIRMM, University of Montpellier, CNRS, France

Abstract

We consider robust combinatorial optimization problems with cost uncertainty where the decision maker can prepare K solutions beforehand and chooses the best of them once the true cost is revealed. Also known as min-max-min robustness (a special case of K-adaptability), it is a viable alternative to otherwise intractable two-stage problems. The uncertainty set assumed in this paper considers that in any scenario, at most Γ of the components of the cost vectors will be higher than expected, which corresponds to the extreme points of the budgeted uncertainty set.

While the classical min-max problem with budgeted uncertainty is essentially as easy as the underlying deterministic problem, it turns out that the min-max-min problem is $\mathcal{NP}$-hard for many easy combinatorial optimization problems, and not approximable in general. We thus present an integer programming formulation for solving the problem through a row-and-column generation algorithm. While exact, this algorithm can only cope with small problems, so we present two additional heuristics leveraging the structure of budgeted uncertainty. We compare our row-and-column generation algorithm and our heuristics on knapsack and shortest path instances previously

Email addresses: marc.goerigk@uni-siegen.de (Marc Goerigk), kurtz@mathc.rwth-aachen.de (Jannis Kurtz), michael.poss@lirmm.fr (Michael Poss)

used in the scientific literature and find that the heuristics obtain good quality solutions in short computational times.

Keywords: robust combinatorial optimization, min-max-min robustness, K-adaptability, budgeted uncertainty.

---

1. Introduction

Let us consider a combinatorial optimization problem

$$\min_{x \in \mathcal{X}} \boldsymbol{c}^\top \boldsymbol{x}, \tag{M^1}$$

where $\mathcal{X} \subseteq \{0,1\}^n$ is the set of feasible solutions and $\boldsymbol{c} \in \mathbb{R}^n$ is a cost vector. In many practical applications (e.g. uncertain road lengths for a shortest path problem or uncertain revenues in a project investment problem), the decisions $\boldsymbol{x}$ must be taken prior to knowing the exact values of the cost vector $\boldsymbol{c}$. In that context, one should account for these uncertainties when looking for a solution. One widely used approach is robust optimization, in which it is assumed that $\boldsymbol{c}$ can take any value in a given uncertainty set $\mathcal{U} \subseteq \mathbb{R}^n$, leading to the robust counterpart

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{c} \in \mathcal{U}} \boldsymbol{c}^\top \boldsymbol{x}. \tag{M^2}$$

The min-max problem (M^2) has been widely studied for discrete uncertainty sets $\mathcal{U} = \{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^m\}$ (see the surveys [ABV09, KZ16]). For most classical combinatorial problems the min-max problem (M^2) is $\mathcal{NP}$-hard for general discrete uncertainty sets $\mathcal{U}$, even if $\mathcal{U}$ only contains two scenarios. What is more, discrete uncertainty sets rely on accurate historical data, which are often not available. When this is the case, it can be more natural for the decision maker to provide only two values $\{\hat{c}_i, \hat{c}_i + d_i\}$ with $d_i \geq 0$ for each uncertain cost component, representing the expected and worst case, respectively. Then, assuming that only Γ uncertain parameters take their

upper values in any scenario, Bertsimas and Sim [BS03] introduced what is often called the discrete budgeted uncertainty set

$$\mathcal{U}^\Gamma = \left\{ \boldsymbol{c} \in \mathbb{R}^n : c_i = \hat{c}_i + \delta_i d_i, i \in [n],\ \boldsymbol{\delta} \in \{0,1\}^n,\ \sum_{i \in [n]} \delta_i \leq \Gamma \right\}$$

where we use the notation $[n] := \{1, \ldots, n\}$. The implicit description of $\mathcal{U}^\Gamma$ leads to min-max problems that are essentially as easy as the underlying deterministic problems (see [BS03]).

In some real-world situations, the min-max approach (M^2) can be too conservative, as no recourse action can be taken to remedy to the values taken by the uncertain parameters. A more flexible model has been introduced in [BK17] to overcome this limitation. The approach computes K solutions and chooses the best of them for each realization of $\boldsymbol{c}$ in $\mathcal{U}$. Using discrete budgeted uncertainty in this setting leads to the min-max-min problem

$$\min_{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)} \in \mathcal{X}} \max_{\boldsymbol{c} \in \mathcal{U}^\Gamma} \min_{k \in [K]} \boldsymbol{c}^\top \boldsymbol{x}^{(k)}. \tag{M^3}$$

Approach (M^3) is particularly well-suited in situations where one can prepare the ground before the uncertainty is revealed. It is a special case of the more general K-adaptability approach from [HKW15], where additional first-stage costs are allowed. Examples are numerous, e.g. transporting relief supplies or evacuating citizens in case an uncertain disaster arises [CTC07, LPdB$^+$13], hub locations [ANSdG12], or parcel deliveries [EKBC19, SGW17]; see also [CGKP19] for more details.

It has been shown in [BK17] that Problem (M^3) is as easy as the underlying problem (M^1) if $\mathcal{U}$ is a convex uncertainty set and $K \geq n+1$. For discrete uncertainty however, the min-max-min problem is at least as complex as the min-max problem which is, more often than not, $\mathcal{NP}$-hard [BK18a]. Regard-

ing the budgeted uncertainty, its convex hull

$$\left\{ \boldsymbol{c} \in \mathbb{R}^n : c_i \in [\hat{c}_i, \hat{c}_i + d_i], i \in [n], \sum_{i \in [n]} \frac{c_i - \hat{c}_i}{d_i} \leq \Gamma \right\} \tag{1}$$

has been considered in [Cha17] and [CGKP19], where the authors study the theoretical complexity of the problem, and propose efficient solution algorithms, respectively.

The focus of this paper is Problem (M^3) under the discrete budgeted uncertainty set $\mathcal{U}^\Gamma$. First, we remark that, unlike the min-max problem, it is not equivalent to replace $\mathcal{U}^\Gamma$ by its convex hull (1), see for instance [BK18a] for an example, so that the results for convex uncertainty sets proved in [BK17, Cha17] do not carry over the problem studied in this paper. Moreover the discrete version of the budgeted uncertainty set is inevitable if we want to model edge failures for problems on graphs or adversarial attacks on labeled data in classification problems, see [BDPZ18] for the latter application.

We prove in Section 2 that, unlike the classical min-max problem, Problem (M^3) for discrete budgeted uncertainty is $\mathcal{NP}$-hard for the shortest path problem, the spanning tree problem, the assignment problem, the knapsack problem, the selection problem and even for the unconstrained minimization problem. We also show that, in general, Problem (M^3) cannot be approximated in polynomial time. On the positive side, we provide dynamic programming algorithms for the unconstrained and the knapsack problem, which are essentially of theoretical interest. Section 3 turns to integer linear programming (ILP) formulations. We first show that computing the inner maximization problem

$$\max_{\boldsymbol{c} \in \mathcal{U}^\Gamma} \min_{k \in [K]} \boldsymbol{c}^\top \boldsymbol{x}^{(k)}$$

is $\mathcal{NP}$-hard, suggesting that no compact ILP may be available for (M^3). We then introduce an assignment-based formulation, that is embedded into a

row-and-column generation algorithm. We pursue numerical approaches for the problem in Section 4 where we provide two heuristic algorithms. Leveraging known results for budgeted uncertainty, we can prove both algorithms solve only polynomially many deterministic problems. Our ILP formulation and the heuristics are numerically assessed in Section 5 on instances of knapsack and shortest path problems previously considered in the literature. Section 6 concludes the paper.

2. Complexity results

In this section, we provide complexity results for the min-max-min problem (M^3) for the unconstrained binary problem (where $\mathcal{X} = \{0,1\}^n$), the selection problem (where $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^n : \sum_{i\in[n]} x_i = p\}$ for an integer $p < n$), the knapsack problem, the spanning tree problem, the assignment problem, and the shortest path problem. Note that for the knapsack problem, we consider a max-min-max problem instead of (M^3). It turns out that all of the mentioned problems are at least weakly $\mathcal{NP}$-hard, and under some configurations even strongly $\mathcal{NP}$-hard. Table 1 summarizes the results which we prove in the following. While the unconstrained binary problem and the selection problem are trivial problems in the nominal case, they are often considered in the robust optimization literature (see, e.g., [BK18b, CGKZ18]). We further show that the shortest path problem becomes inapproximable, and provide a dynamic programming algorithm for the knapsack problem.

2.1. $\mathcal{NP}$-hard cases and inapproximability

In the following we prove $\mathcal{NP}$-hardness results for Problem (M^3) for several combinatorial problems.

Problem	K fixed	K input	Reference
Unconstrained	weakly $\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	Theorem 1
Selection	weakly $\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	Theorem 2
Knapsack	weakly $\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	Theorem 3
Spanning Tree	$\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	Theorem 3
Assignment	$\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	Theorem 3
Shortest Path	strongly $\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	Theorem 4

Table 1: Complexity of Problem (M^3) for $\mathcal{U}^\Gamma$.

Theorem 1. *Problem* (M^3) *for the unconstrained binary problem is weakly $\mathcal{NP}$-hard, even if $K = 2$. It is strongly $\mathcal{NP}$-hard when K is part of the input.*

Proof. First assume that $\Gamma = 1$ and $K = 2$ and consider a set of integers $a_i \in \mathbb{N}$, $i \in [n]$. The partition problem, i.e. deciding whether there exists a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = \sum_{i \in [n] \setminus S} a_i$, is known to be $\mathcal{NP}$-hard. Next, consider an instance of problem (M^3) where $\hat{c}_i = -a_i$ and $d_i = M$ for each $i \in [n]$ and $M = \sum_{i \in [n]} a_i$. By the choice of M in each optimal solution it holds $\boldsymbol{x}_i^{(1)} \neq \boldsymbol{x}_i^{(2)}$ for all $i \in [n]$ and the large deviation M will affect the solution with the smaller nominal costs. The optimal value is therefore larger or equal to $-\frac{1}{2}M$. Then, the partition instance is a yes instance if and only if the min-max-min problem has an optimal value equal to $-\frac{1}{2}M$.

Since the unconstrained binary problem is a special case of the knapsack problem the pseudo-polynomial algorithm presented in Section 2.2 can be applied to (M^3) for the unconstrained problem, which proves the weak $\mathcal{NP}$-hardness.

The proof above extends to the case when K is part of the input by setting $\Gamma = K - 1$. A similar reasoning shows that one can decide if a partition into K sets having the same costs exists if and only if the corresponding min-max-min problem has an optimal value equal to $-\frac{1}{K}M$. The problem generalizes

the 3-partition problem (see below), which is $\mathcal{NP}$-hard in the strong sense, which proves the result. □

Theorem 2. *The problem* (M^3) *is weakly $\mathcal{NP}$-hard for the selection problem, even if $K = 2$. It is strongly $\mathcal{NP}$-hard if K is part of the input.*

Proof. For $K = 2$ we can use a similar construction as in the proof of Theorem 1 with $p = n/2$, as the partition problems remains (weakly) $\mathcal{NP}$-hard if both partitions are required to have equal size. Note that negative cost vectors are not required in this case. Moreover, one can adapt the pseudo-polynomial algorithm presented in Section 2.2 to the selection problem which proves the weakly $\mathcal{NP}$-hardness.

To prove strong $\mathcal{NP}$-hardness for the case that K is part of the input we reduce the 3-partition problem to Problem (M^3). The 3-partition problem is defined as follows: For given values $c_1, \ldots, c_{3m} \in \mathbb{N}$ and a bound $B \in \mathbb{N}$ such that $\frac{B}{4} < c_i < \frac{B}{2}$ for all $i = 1, \ldots, 3m$ and $\sum_{i=1}^{3m} c_i = mB$ we have to decide if there exist subsets $A_1, \ldots, A_m \subseteq \{1, \ldots, 3m\}$ such that $\sum_{i \in A_j} c_i = B$ for all $j = 1 \ldots, m$. Note that each set A_j must contain exactly 3 elements. The 3-partition problem is known to be strongly $\mathcal{NP}$-hard [GJ90].

We assume we have a given instance of the 3-partition problem as above. We now define an instance of the min-max-min selection problem in dimension $3m$ where $p = 3$, $K = m$, $\Gamma = K - 1$ and we define $\mathcal{U}^\Gamma$ with $\hat{c}_i = c_i$ and $d_i = M$ where $M = \sum_{i=1}^{3m} c_i$. By the definition of M and Γ the K solutions define a partition of the ground set $\{1, \ldots, 3m\}$ in an optimal solution of (M^3), and each solution uses exactly 3 elements. The given 3-partition is then a yes instance if and only if Problem (M^3) has an optimal value lower or equal to B.

□

Theorem 3. *Even if $K = 2$, Problem* (M^3) *for the spanning tree problem and the assignment problem is $\mathcal{NP}$-hard; for the knapsack problem it is weakly $\mathcal{NP}$-hard. It is strongly $\mathcal{NP}$-hard for all the mentioned problems if K is part of the input.*

The proof of Theorem 3 follows the line of the proof of Theorem 1 and is provided in Appendix A. The result for the knapsack problem directly follows from Theorem 1 by choosing knapsack weights and a knapsack capacity such that all vectors in $\{0,1\}^n$ are feasible.

Theorem 4. *Problem* (M^3) *for the shortest path problem is strongly $\mathcal{NP}$-hard, even if $K = 2$.*

Proof. Assume that $\Gamma = 1$ and $K = 2$ and consider a weighted undirected graph $G = (V, E, \boldsymbol{w})$ and two nodes s and t in V. It is known that finding two edge-disjoint paths between s and t such that the length of the longer path is minimized is $\mathcal{NP}$-hard in the strong sense [LMSL90]. Next, consider an instance of problem (M^3) where we need to find paths between s and t in G and define $\hat{c}_e = w_e$ and $d_e = M$ for each $e \in E$, where M can be chosen as $M = \sum_{e \in E} \hat{c}_e$. Clearly there exist two disjoint paths in G where the longer path has costs lower or equal to some value L if and only if the latter instance of the min-max-min problem has an optimal value lower or equal to L.

□

Our last complexity result shows that the problem may not be approximable even if the underlying problem is polynomially solvable.

Theorem 5. *Problem* (M^3) *for the shortest path problem where paths may have up to 5 edges cannot be approximated in polynomial time if K is part of the input unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Consider a weighted undirected graph $G = (V, E, \boldsymbol{w})$ and two nodes s and t in V. It is known that determining whether there exist K edge-disjoint paths between s and t having at most 5 edges is $\mathcal{NP}$-hard [IPS82]. By defining $\Gamma = K - 1$, $\hat{\boldsymbol{c}} = \mathbf{0}$ and $\boldsymbol{d} = \mathbf{1}$, the optimal solution cost of (M^3) is 0 if and only if the answer to the decision problem is yes. □

2.2. Dynamic programming for the knapsack problem

We provide a dynamic programming algorithm for the knapsack version of problem (M^3), which runs in pseudo-polynomial time. As the complexity of the algorithm is high, its purpose is essentially theoretical. In what follows we suppose the feasibility set is given by

$$\mathcal{X}_{KP} = \left\{ \boldsymbol{x} \in \{0,1\}^n : \sum_{i \in [n]} w_i x_i \leq C \right\}$$

where w_i denotes the weight of item i and C the total available capacity. Our algorithm runs in pseudo-polynomial time if K and Γ are fixed, and has an exponential running-time in general, in accordance with the complexity stated in Theorem 1. The algorithm can easily be adapted to handle the unconstrained binary problem by choosing knapsack weights and a capacity such that all binary vectors are feasible and transforming the problem into a minimization problem. For the selection problem, the dynamic programming algorithm extends by enforcing that the capacity constraint be satisfied at equality when computing the costs in step 17 of Algorithm 1.

Algorithm 1 follows the classical dynamic programming algorithm for scheduling jobs on unrelated machines proposed in [HS76] by iterating over all items and creating labels $\boldsymbol{s} \in \mathcal{S}$ for each possible variant of adding the item to the feasible solutions or not. For ease of notation we present the idea of the algorithm for the case $K = 2$ and $\Gamma = 1$, and sketch the generalization in Appendix B.

Algorithm 1: Solving the robust knapsack problem for $K = 2$ and $\Gamma = 1$

Input: $\mathcal{X}_{KP}$, $\mathcal{U}^{\Gamma}$, $K = 2$, $\Gamma = 1$
$\mathcal{S} = \{(0, 0, 0, 0, 0, 0, 0)\}$
for $i \in [n]$ **do**
  for $(w^{(1)}, w^{(2)}, c^{(1)}, c^{(2)}, d^{(1)}, d^{(2)}, d^{(1,2)}) \in \mathcal{S}$ **do**
    if $w^{(1)} + w_i \leq C$ **then**
      $\boldsymbol{s}^{(1)} = (w^{(1)} + w_i, w^{(2)}, c^{(1)} + \hat{c}_i, c^{(2)}, \max\left(d^{(1)}, d_i\right), d^{(2)}, d^{(1,2)})$
      if $w^{(2)} + w_i \leq C$ **then**
        $\boldsymbol{s}^{(1,2)} = (w^{(1)} + w_i, w^{(2)} + w_i, c^{(1)} + \hat{c}_i, c^{(2)} + \hat{c}_i, \max\left(d^{(1)}, d_i\right), \max\left(d^{(2)}, d_i\right), \max\left(d^{(1,2)}, d_i\right))$
      end
    end
    else if $w^{(2)} + w_i \leq C$ **then**
      $\boldsymbol{s}^{(2)} = (w^{(1)}, w^{(2)} + w_i, c^{(1)}, c^{(2)} + \hat{c}_i, d^{(1)}, \max\left(d^{(2)}, d_i\right), d^{(1,2)})$
    end
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \boldsymbol{s}^{(1,2)}\}$
  end
end
Compute $\boldsymbol{s}_{\max} \in \mathcal{S}$ with maximal $cost(\boldsymbol{s})$ (see Eq. (2))
Output: $\boldsymbol{s}_{\max}$

For each item $i \in [n]$ we assume that the current set of labels $\mathcal{S}$ has been constructed by deciding whether the first $i-1$ items are added to (partial) solution $\boldsymbol{x}^{(1)} \in \mathcal{X}_{KP}$, $\boldsymbol{x}^{(2)} \in \mathcal{X}_{KP}$ or to both. Hence, the partial solutions correspond to sets of items $S^{(1)} \subseteq [i-1]$ and $S^{(2)} \subseteq [i-1]$, respectively. As indexing the label $\boldsymbol{s}$ by $S^{(1)}$ and $S^{(2)}$ could possibly lead to exponentially many labels, we instead define the label as

$$\boldsymbol{s} = (w^{(1)}, w^{(2)}, c^{(1)}, c^{(2)}, d^{(1)}, d^{(2)}, d^{(1,2)}),$$

where $w^{(k)} = \sum_{i \in S^{(k)}} w_i$, $c^{(k)} = \sum_{i \in S^{(k)}} c_i$, and $d^{(k)} = \max_{i \in S^{(k)}} d_i$ for $k \in \{1, 2\}$. The value $d^{(1,2)}$ represents the highest deviations among the items included in both solutions up to now, that is, $d^{(1,2)} = \max_{i \in S^{(1)} \cap S^{(2)}} d_i$. Next,

we have four possibilities concerning the addition of item i to the partial solutions:

1. $S^{(1)} \leftarrow S^{(1)} \cup \{i\}$, yielding the new label $\boldsymbol{s}^{(1)}$, see step 6
2. $S^{(2)} \leftarrow S^{(2)} \cup \{i\}$, yielding the new label $\boldsymbol{s}^{(2)}$, see step 12
3. $S^{(1)} \leftarrow S^{(1)} \cup \{i\}$ and $S^{(2)} \leftarrow S^{(2)} \cup \{i\}$, yielding the new label $\boldsymbol{s}^{(1,2)}$, see step 8.
4. item i is added to none of the two solutions, which does not change the current label.

Notice that the first three possibilities occur only if the resulting partial solutions do not exceed the capacity of the knapsack.

Finally, we detail how to calculate the costs of a label, i.e. the worst-case over all scenarios in $\mathcal{U}^\Gamma$ for the corresponding solution. Note that since $\Gamma = 1$ that cost can be computed by examining the following three possibilities: either we allow a deviation on the item with the largest deviation d_i contained in the first solution or contained in the second solution or in both solutions. Therefore the worst-case costs can be calculated by

$$\begin{aligned} cost(\boldsymbol{s}) = \max \Big\{ & \min(c^{(1)} + d^{(1)}, c^{(2)}), \\ & \min(c^{(1)} + d^{(1,2)}, c^{(2)} + d^{(1,2)}), \\ & \min(c^{(1)}, c^{(2)} + d^{(2)}) \Big\}. \end{aligned} \tag{2}$$

To investigate the run-time of the algorithm consider $\overline{c} = \max_{i\in[n]} \hat{c}_i$, $\overline{d} = \max_{i\in[n]} d_i$ and $\overline{w} = \max_{i\in[n]} w_i$. Then the largest value we have to consider for $c^{(1)}, c^{(2)}$ is $n\overline{c}$, the largest value for $w^{(1)}, w^{(2)}$ is $n\overline{w}$ and the largest value for $d^{(1)}, d^{(2)}, d^{(1,2)}$ is $\overline{d}$. Therefore the number of different labels we have to consider in the algorithm is at most $n^4\overline{c}^2\overline{w}^2\overline{d}^3$, so that the running time of

Algorithm 1 is in $O(n^5\overline{c}^2\overline{w}^2\overline{d}^3)$. Since we may record the indices that deviate instead of the deviations themselves, another valid bound for the running time of the algorithm is $O(n^8\overline{c}^2\overline{w}^2)$. Specifically, we could define any label in $\mathcal{S}$ as the 7-tuple $\boldsymbol{s}' = (w^{(1)}, w^{(2)}, c^{(1)}, c^{(2)}, i^{(1)}, i^{(2)}, i^{(1,2)})$ where $i^{(1)}, i^{(2)}$ and $i^{(1,2)}$ belong to $[n]$.

Many dynamic programming algorithms lead to approximation algorithms that can provide $(1+\epsilon)$-approximate solutions with a complexity that is polynomial in the input of the problem and $1/\epsilon$. We prove below that this is not the case here, by adapting the reduction from the equipartition problem to the knapsack problem from [KPP04].

Theorem 6. *There is no FPTAS for the knapsack variant of problem* (M^3) *unless* $\mathcal{P} = \mathcal{NP}$*, even in the case* $\Gamma = 1$*.*

Proof. We reduce the equipartition problem to the decision version of (M^3) for $\Gamma = 1$ and $K = 2$. Given an instance of the equipartition problem i.e. $a_i \in \mathbb{N}$ for each $i \in N = [n]$ where n is even, the equipartition problem is to decide if there exists an index set $I \subset N$ with $|I| = \frac{n}{2}$ such that $\sum_{i\in I} a_i = \sum_{i\in N\setminus I} a_i$. This problem is known to be $\mathcal{NP}$-hard [GJ90] . We define a knapsack instance as follows: for each $i \in N$ we set $w_i = a_i$, $\hat{c}_i = 1$ and $d_i = -n$. The capacity is set to $C = \frac{1}{2}\sum_{i\in N} a_i$. Clearly, there exists an equipartition of the elements of N if and only if there exists a solution of Problem (M^3) for the knapsack instance with profit at least $\frac{n}{2}$. This yields the result, as an FPTAS would compute an $\frac{1}{n+1}$-approximate solution in polynomial time, which would be optimal for that instance. □

Note that while it is not possible to construct an FPTAS for the robust knapsack problem, it stands to reason that the dynamic programming approach presented in this section can be used to construct a PTAS based on cost inflation.

3. Exact algorithm and lower bounds

We first prove that even evaluating the objective function of Problem (M^3), i.e., calculating

$$cost(\mathbf{x}) := \max_{\boldsymbol{c}\in\mathcal{U}^\Gamma} \min_{k\in[K]} \boldsymbol{c}^\top \boldsymbol{x}^{(k)}$$

for fixed $\mathbf{x} = (\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)})$ is strongly $\mathcal{NP}$-hard. This makes it unlikely that a compact ILP formulation for Problem (M^3) exists.

Theorem 7. *Evaluating the objective function of Problem* (M^3) *for a given solution is strongly $\mathcal{NP}$-hard.*

Proof. Given an integer N and a collection $\mathcal{S}$ of m sets $S_i \subseteq [N]$, the set cover problem looks for a sub-collection of $\mathcal{S}$ of cardinality not greater than L and whose union equals $[N]$. We construct a reduction through the following instance of $cost(\mathbf{x})$. We set $K = N$, $\Gamma = L$, $n = m$, $\hat{c}_i = 0$ and $d_i = 1$ for each $i \in [n]$. Further, we define $x_i^{(k)} = 1$ iff $k \in S_i$. Clearly, $cost(\mathbf{x}) \geq 0$. We prove next that $cost(\mathbf{x}) \geq 1$ if and only if the answer to the set cover instance is *yes*.

Let us define $\mathcal{Z} = \{\boldsymbol{z} \in \{0,1\}^n : \sum_{i\in[n]} z_i \leq \Gamma\}$ as the set of vectors $\boldsymbol{z}$ describing costs $\boldsymbol{c} = (\hat{c}_i + d_i z_i)_{i\in[n]} \in \mathcal{U}^\Gamma$. There exists a bijection between the elements of $\mathcal{Z}$ and the sub-collections of $\mathcal{S}$ of cardinality not greater than Γ. Further, we see that $\boldsymbol{c}^\top \boldsymbol{x}^{(k)} = \sum_{i\in[n]} z_i x_i^{(k)}$, so setting $z_i = 1$ implies that $\sum_{i\in[n]} z_i x_i^{(k)} \geq 1$ for each $k \in S_i$. From the definition of $cost(\mathbf{x})$, we have that

$$cost(\mathbf{x}) \geq 1 \Leftrightarrow \exists \boldsymbol{z} \in \mathcal{Z}, \forall k \in [K] : \sum_{i\in[n]} z_i x_i^{(k)} \geq 1. \tag{3}$$

If the answer to the set cover problem is *yes* and is provided by the sub-collection indexed by $\mathcal{M}$, then we set $z_i = 1$ for each $i \in \mathcal{M}$ and obtain from (3) that $cost(\mathbf{x}) \geq 1$. If the answer is *no*, then for each $\boldsymbol{z} \in \mathcal{Z}$, there exists $k \in [K]$ such that $\sum_{i\in[n]} z_i x_i^{(k)} = 0$, and we obtain $cost(\mathbf{x}) = 0$. □

In the following we provide an ILP formulation for Problem (M^3) and derive a row-and-column generation algorithm based on this formulation. Consider first an arbitrary discrete uncertainty set $\mathcal{U} = \{\boldsymbol{c}^1, \ldots, \boldsymbol{c}^m\}$ and let binary variables y_{kj} define an assignment between the scenarios and the solutions, where $y_{kj} = 1$ if and only if $\boldsymbol{x}^{(k)}$ has the minimal objective value over all $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)}$ in scenario $\boldsymbol{c}^j$. We can reformulate Problem (M^3) as

$$
\begin{array}{lll}
\min & \omega & \\
\text{s.t.} & \omega \geq \displaystyle\sum_{k\in[K]} y_{kj} \left(\sum_{i\in[n]} c_i^j x_i^{(k)} \right) & \forall j \in [m] \\
& \displaystyle\sum_{k\in[K]} y_{kj} = 1 & \forall j \in [m] \\
& y_{kj} \in \{0,1\} & \forall k \in [K],\ j \in [m] \\
& \boldsymbol{x}^{(k)} \in \mathcal{X} & \forall k \in [K],
\end{array}
\tag{Master}
$$

where the product $y_{kj} x_i^{(k)}$ can be linearized introducing the additional variables $w_{kji} = y_{kj} x_i^{(k)}$ and rewriting (Master) to

$$
\begin{array}{lll}
\min & \omega & \\
\text{s.t.} & \omega \geq \displaystyle\sum_{k\in[K]} \sum_{i\in[n]} c_i^j w_{kji} & \forall j \in [m] \\
& \displaystyle\sum_{k\in[K]} y_{kj} = 1 & \forall j \in [m] \\
& w_{kji} \geq x_i^{(k)} + y_{kj} - 1 & \forall k \in [K], j \in [m], i \in [n] \\
& y_{kj} \in \{0,1\} & \forall k \in [K],\ j \in [m] \\
& w_{kji} \geq 0 & \forall k \in [K], j \in [m], i \in [n] \\
& \boldsymbol{x}^{(k)} \in \mathcal{X} & \forall k \in [K].
\end{array}
\tag{4}
$$

Note that (Master) has exponentially many variables and constraints in case of discrete budgeted uncertainty. The first ingredient of our approach, described in Algorithm 2, is to solve (Master) for a starting set $\mathcal{U}' \subset \mathcal{U}^\Gamma$ and

to iteratively add a new scenario which is the optimal solution of problem

$$\max_{\boldsymbol{c}\in\mathcal{U}^\Gamma} \min_{k\in[K]} \boldsymbol{c}^\top \boldsymbol{x}^{(k)} \tag{Slave}$$

for the current solution $\mathbf{x}$ to the restricted master problem. Note that the optimal value of the latter problem is the objective value $cost(\mathbf{x})$ of $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)}$ for problem (M^3), which can be computed through the following IP formulation:

$$\begin{aligned} \max \quad & z \\ s.t. \quad & z \leq \hat{c}^\top \boldsymbol{x}^{(k)} + \sum_{i=1}^{n} \delta_i d_i \boldsymbol{x}_i^{(k)} \;\; k \in [K] \\ & \sum_{i=1}^{n} \delta_i \leq \Gamma \\ & \delta_i \in \{0,1\} \;\; i \in [n]. \end{aligned} \tag{5}$$

Clearly the optimal value of (Master) for a subset of scenarios is a lower bound for Problem (M^3) while the optimal value of (Slave) is an upper bound. Therefore Algorithm 2 iteratively calculates upper and lower bounds with decreasing gap. A similar idea for robust two-stage problems was already presented in [ZZ13].

Algorithm 2 calculates an optimal solution of Problem (M^3). Since there is a finite number of feasible solutions, we can only generate a finite number of scenarios in the loop and therefore the algorithm terminates in a finite number of steps.

As mentioned above, Algorithm 2 iteratively calculates a non-decreasing sequence of lower bounds for Problem (M^3). Nevertheless as our computational experiments show these lower bounds are hard to compute and tend to be far from the optimal value even after one hour of computation time; see Section 5. In the following we present a different lower bound for Problem (M^3) which turns out to be tighter as well as easier to compute.

Algorithm 2: Row-and-column generation algorithm

Input: $\mathcal{X}$, $\mathcal{U}^\Gamma$, K
$\mathcal{U}' \leftarrow \emptyset$
Choose any $\boldsymbol{c} \in \mathcal{U}^\Gamma$
repeat
  $\mathcal{U}' \leftarrow \mathcal{U}' \cup \{\boldsymbol{c}\}$
  Solve (Master) with respect to $\mathcal{U}'$
  Set $\mathbf{x}$ as its optimal solution, LB as its objective value
  Solve (Slave) with respect to $\mathbf{x}$
  Set $\boldsymbol{c}$ as its optimal solution, UB as its objective value
until $LB = UB$;
Output: $\mathbf{x}$

Observation 8. *The optimal value of problem*

$$\max_{\boldsymbol{c}\in\mathcal{U}^\Gamma} \min_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{c}^\top \boldsymbol{x} \tag{MMLB}$$

is a lower bound for Problem (M^3).

Proof. Clearly for every solution $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)}$ we have

$$\max_{\boldsymbol{c}\in\mathcal{U}^\Gamma} \min_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{c}^\top \boldsymbol{x} \leq \max_{\boldsymbol{c}\in\mathcal{U}^\Gamma} \min_{k\in[K]} \boldsymbol{c}^\top \boldsymbol{x}^{(k)}$$

which proves the result. □

The problem (MMLB) can be solved by a classical row-generation method as follows: For a subset of solutions $\mathcal{X}' \subset \mathcal{X}$ calculate an optimal solution $(\boldsymbol{c}^*, z^*)$ of problem

$$\begin{aligned} &\max\ z \\ &\text{s.t. } z \leq \boldsymbol{c}^\top \boldsymbol{x} && \forall \boldsymbol{x} \in \mathcal{X}' \\ &\quad \boldsymbol{c} \in \mathcal{U}^\Gamma \end{aligned}$$

and afterwards solve the deterministic problem

$$\min_{\boldsymbol{x}\in\mathcal{X}} (\boldsymbol{c}^*)^\top \boldsymbol{x}.$$

Add the optimal solution of the latter problem to $\mathcal{X}'$ and iterate. Stop if the latter optimal value is larger than or equal to z^*.

4. Heuristic algorithms

In this section we present two heuristic algorithms which are based on the idea to find a partition of the uncertainty set into K subsets and calculate the optimal min-max solution for each of the subsets, see the general scheme presented in Algorithm 3. To end up with a fast algorithm the min-max problem for each subset should be computationally tractable. For both heuristics we derive a K-partition of the budgeted uncertainty set such that each subset remains a budgeted uncertainty set or has a structure which is close to a budgeted uncertainty set. In both cases we can show that each of the min-max problems in Algorithm 3 can be solved by solving a polynomial number of deterministic problems (M^1).

Algorithm 3: Heuristic Algorithm for Problem (M^3) with $K \leq n$.

1 **Input:** $\mathcal{X}$, $\mathcal{U}^\Gamma$, K
2 Calculate a partition $\mathcal{U}_1 \cup \ldots \cup \mathcal{U}_K = \mathcal{U}^\Gamma$.
3 Calculate

$$\boldsymbol{x}^{(k)} = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}_k} \boldsymbol{c}^\top \boldsymbol{x} \qquad \forall k \in [K].$$

4 **Output:** $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)}$

4.1. Heuristic 1

In this section we derive a partition of the budgeted uncertainty set $\mathcal{U}^\Gamma$ such that each of the subsets has a similar structure as the classical budgeted

uncertainty set. Furthermore each subset of the partition covers scenarios which are close to each other in the sense that there exists a solution $\boldsymbol{x} \in \mathcal{X}$ which works well for most of the scenarios. We show that for each subset of the partition the min-max problem can be solved by solving a polynomial number of deterministic problems.

We partition the budgeted uncertainty set such that in each subset $\mathcal{U}_k$ a subsequence of items is selected and in each scenario of the subset at least one of the items deviates from its mean value. More precisely, let any ordering of the indices $i_1, \ldots, i_n$ be given. Without loss of generality, we assume $i_\ell = \ell$ in the following presentation. For $t := \lfloor \frac{n}{K} \rfloor$, we define

$$\mathcal{U}_k := \Bigg\{ \boldsymbol{c} \in \mathbb{R}^n \ : c_i = \hat{c}_i + \delta_i d_i,\ \boldsymbol{\delta} \in \{0,1\}^n, \sum_{i=1}^{(k-1)t} \delta_i = 0, \sum_{i=(k-1)t+1}^{kt} \delta_i \geq 1, \sum_{i=(k-1)t+1}^{n} \delta_i \leq \Gamma \Bigg\} \tag{6}$$

for each $k = 1, \ldots, K-1$ and

$$\mathcal{U}_K := \Bigg\{ \boldsymbol{c} \in \mathbb{R}^n \ : c_i = \hat{c}_i + \delta_i d_i,\ \boldsymbol{\delta} \in \{0,1\}^n, \sum_{i=1}^{(K-1)t} \delta_i = 0, \sum_{i=(K-1)t+1}^{n} \delta_i \geq 1, \sum_{i=(K-1)t+1}^{n} \delta_i \leq \Gamma \Bigg\}.$$

Note that the only difference in the definition of $\mathcal{U}_K$ is that the second sum instead of containing t summands, additionally contains all summands which are left due to rounding of t. For ease of notation we do not consider this special case in the following.

It is easy to see that $\mathcal{U}_1 \cup \cdots \cup \mathcal{U}_K = \mathcal{U}^\Gamma \backslash \{\hat{c}\}$. Furthermore all of the subsets $\mathcal{U}_k$ have a budgeted-like structure. We will use this structure in the following theorem to show that the classical min-max problem in Step 3 of Algorithm

3 can be solved in polynomial time if an oracle for the underlying deterministic problem is given. We define in the following $(x)_+ := \max\{x, 0\}$. The following result is related to Theorem 3 from [BS03] and its generalizations in [Pos18].

Theorem 9. *The min-max problem with uncertainty set $\mathcal{U}_k$ can be solved by solving the deterministic problems*

$$\alpha^*\Gamma - \beta^* + t(\beta^* - \alpha^*)_+ + \min_{\boldsymbol{x}\in\mathcal{X}} \hat{\boldsymbol{c}}^\top \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{x} \tag{7}$$

where

$$w_i = \begin{cases} 0 & \text{if } i \leq (k-1)t \\ (d_i + \beta^* - \alpha^*)_+ - (\beta^* - \alpha^*)_+ & \text{if } (k-1)t+1 \leq i \leq kt \\ (d_i - \alpha^*)_+ & \text{if } i \geq kt+1 \end{cases}$$

for all values

$$\begin{aligned}(\alpha^*, \beta^*) \in A \times \{0\} \cup \{(\alpha, \beta) \mid \alpha \in A, \beta = \alpha\} \\ \cup \{(\alpha, \beta) \mid \alpha \in A, \beta = (\alpha - d_i)_+ : \ i = (k-1)t+1, \ldots, kt\},\end{aligned}$$

where $A := \{d_i \mid i = (k-1)t+1, \ldots, n\} \cup \{0\}$ and returning the solution of the problem with the smallest optimal value.

Proof. For each given $\boldsymbol{x} \in \mathcal{X}$ we can rewrite the objective value $\max_{\boldsymbol{c}\in\mathcal{U}_k} \boldsymbol{c}^\top \boldsymbol{x}$ by

$$\begin{aligned} \hat{\boldsymbol{c}}^\top \boldsymbol{x} + \max_{\boldsymbol{\delta}} \quad & \sum_{i=(k-1)t+1}^{n} \delta_i d_i x_i \\ \text{s.t.} \quad & \sum_{i=(k-1)t+1}^{kt} \delta_i \geq 1 \\ & \sum_{i=(k-1)t+1}^{n} \delta_i \leq \Gamma \\ & \delta_i \in [0,1] \quad i = (k-1)t+1, \ldots, n. \end{aligned}$$

The dual of the above linear program is

$$\begin{aligned}\hat{\boldsymbol{c}}^\top \boldsymbol{x} + \min\ & \alpha\Gamma - \beta + \sum_{i=(k-1)t+1}^{n} \gamma_i \\ \text{s.t. } & \alpha - \beta + \gamma_i \geq d_i x_i \quad i = (k-1)t+1, \ldots, kt \\ & \alpha + \gamma_i \geq d_i x_i \quad i = kt+1, \ldots, n \\ & \alpha, \beta \geq 0 \\ & \gamma_i \geq 0 \quad i = (k-1)t+1, \ldots, n.\end{aligned} \tag{8}$$

In each optimal solution of the latter problem for γ_i it holds

$$\gamma_i = (d_i x_i + \beta - \alpha)_+ = x_i(d_i + \beta - \alpha)_+ + (1 - x_i)(\beta - \alpha)_+$$

for each $i = (k-1)t+1, \ldots, kt$ and

$$\gamma_i = (d_i x_i - \alpha)_+ = x_i(d_i - \alpha)_+$$

for each $i = kt+1, \ldots, n$. Therefore, substituting the latter equations in the objective function of Problem (8) we obtain that problem

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{c} \in \mathcal{U}_k} \boldsymbol{c}^\top \boldsymbol{x}$$

is equivalent to

$$\begin{aligned}\min\ & \hat{\boldsymbol{c}}^\top \boldsymbol{x} + \alpha\Gamma - \beta + \sum_{i=(k-1)t+1}^{kt} x_i(d_i + \beta - \alpha)_+ + (1 - x_i)(\beta - \alpha)_+ \\ & + \sum_{i=kt+1}^{n} x_i(d_i - \alpha)_+ \\ \text{s.t. } & \boldsymbol{x} \in \mathcal{X}, \alpha, \beta \geq 0.\end{aligned} \tag{9}$$

For any fixed $\boldsymbol{x}^*$ the objective function of the latter problem is a piecewise linear and convex function, so its minimum is reached at one of its kinkpoints. More specifically, the function is the sum of an affine function

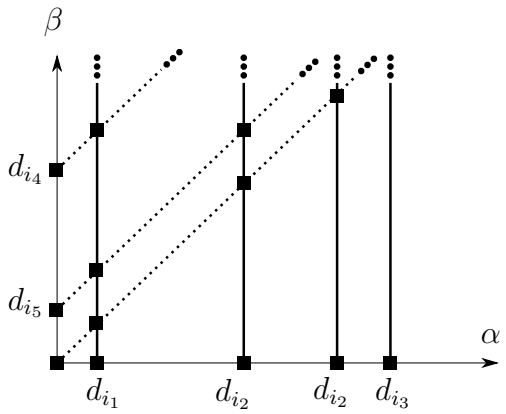

Figure 1: Non-differentiable regions of $f_1(\alpha, \beta)$ and $f_2(\alpha, \beta)$, respectively drawn as dotted and solid lines. The squares represent the kinkpoints of the objective function on the domain visible in the figure.

and two convex piece-wise linear functions, $f_1(\alpha, \beta) = \sum_{i=(k-1)t+1}^{kt} x_i(d_i + \beta - \alpha)_+ + (1 - x_i)(\beta - \alpha)_+$ and $f_2(\alpha, \beta) = \sum_{i=kt+1}^{n} x_i(d_i - \alpha)_+$. The non-differentiable regions of both functions are half-lines, parallel to the lines $\alpha - \beta = 0$ and $\alpha = 0$, respectively, see Figure 1 for an illustration. Therefore, any kinkpoint of the objective function is obtained at the intersection of these lines, proving the result.

□

Corollary 10. *The heuristic presented in Algorithm 3 for the partition given in* (6) *requires the solution of* $\mathcal{O}(K(2+t)n)$ *many deterministic problems.*

Proof. For each of the K subsets $\mathcal{U}_k$ the number of (α, β) values for which we have to solve the deterministic problem is in $\mathcal{O}(2n + tn)$. Since we have to solve the min-max problem for each of the K subsets in Algorithm 3 in total we have to solve $\mathcal{O}(K(2+t)n)$ deterministic problems. □

4.2. Heuristic 2

In this section we present a second heuristic for Problem (M^3), which is also based on partitioning the set $\mathcal{U}$ into K sets $\mathcal{U}_1 \cup \ldots \cup \mathcal{U}_K$, such that the

min-max problem on each set can be solved in polynomial time. Differently from the previous approach, we find the partition dynamically.

Consider again the uncertainty set

$$\mathcal{U}^\Gamma = \left\{ \boldsymbol{c} \in \mathbb{R}^n : c_i = \hat{c}_i + d_i\delta_i,\ \boldsymbol{\delta} \in \{0,1\}^n,\ \sum_{i\in[n]} \delta_i \leq \Gamma \right\}$$

and let us assume that for a specific item $i \in [n]$, we enforce $\delta_i = 1$. We denote the resulting uncertainty set as $\mathcal{U}^{+i}$. It is also possible to enforce $\delta_i = 0$, in which case the resulting set is denoted as $\mathcal{U}^{-i}$. Note that $\mathcal{U}^{+i} \cap \mathcal{U}^{-i} = \emptyset$ and $\mathcal{U}^{+i} \cup \mathcal{U}^{-i} = \mathcal{U}^\Gamma$. We can repeat this branching step on the resulting subsets, until we have constructed a partition consisting of K sets.

This requires two rules: one rule to decide on which of the current sets to branch, and another rule to decide which variable to fix. This is similar to a branch-and-bound method, where we need to decide a node selection and a variable selection policy.

We propose the following rules. For branching, we choose a set for which less than Γ many items are already fixed to 1 (otherwise it consists of a single scenario) and the min-max problem has the highest objective value. This is a greedy choice by which we can hope to reduce the objective value of the current solution. For variable selection, we choose an item $i \in [n]$ that is not yet fixed in the current set, is used in the corresponding min-max solution, and has the highest deviation d_i. This way, we branch on what is estimated to be the current most important item.

Note that after fixing some δ_i variables to be either 0 or 1, the resulting uncertainty set is a classical budgeted uncertainty set and applying Theorem 3 from [BS03], the resulting min-max problem can be solved by solving $\mathcal{O}(n)$ many deterministic problems. With every branching, we need to solve two new such subproblems. To complete, the heuristic thus requires the

solution of $\mathcal{O}(Kn)$ many deterministic problems, and runs overall in polynomial time if the deterministic problem can be solved in polynomial time. Algorithm 4 summarizes this procedure.

Algorithm 4: Heuristic Algorithm for Problem (M^3) with $K \leq n$.

Input: $\mathcal{X}$, $\mathcal{U}^\Gamma$, K
$\mathcal{L} \leftarrow \{\mathcal{U}^\Gamma\}$
repeat
  $\mathcal{U} \leftarrow \arg\max\{\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \boldsymbol{c}^\top \boldsymbol{x} : \mathcal{U} \in \mathcal{L} \text{ not fixed completely}\}$
  $\boldsymbol{x} \leftarrow \min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \boldsymbol{c}^\top \boldsymbol{x}$
  $i \leftarrow \arg\max_{i\in[n]}\{d_i : x_i = 1,\ i \text{ not yet fixed in } \mathcal{U}\}$
  $\mathcal{L} \leftarrow (\mathcal{L} \cup \{\mathcal{U}^{+i}, \mathcal{U}^{-i}\}) \setminus \{\mathcal{U}\}$
until $|\mathcal{L}| = K$;
Output: $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)}$ as minimizers for each $\mathcal{U} \in \mathcal{L}$

We give an example for the approach with $K = 3$ in Figure 2. Here, we

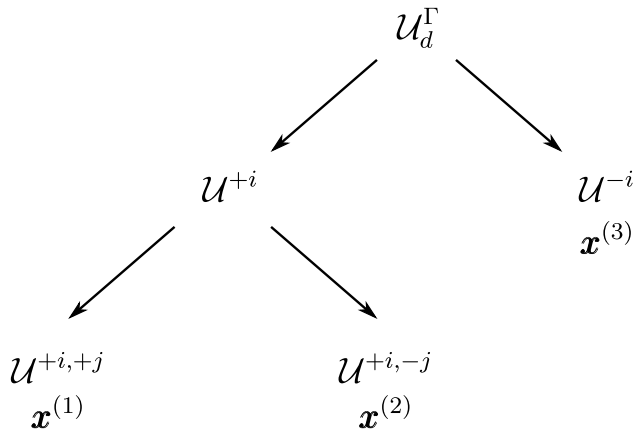

Figure 2: Example for Heuristic 2.

first solve the min-max problem using the original set $\mathcal{U}^\Gamma$. In the resulting solution, let i be the item with the highest deviation d_i. We branch by removing $\mathcal{U}^\Gamma$ from our current list of uncertainty sets, and instead consider

$\mathcal{U}^{+i}$ and $\mathcal{U}^{-i}$. We solve the min-max problem on each. Now let us assume that the resulting objective value is larger on set $\mathcal{U}^{+i}$. We choose this set for our next branching. Let $j \neq i$ be the item in the corresponding solution with largest deviation value d_j. We remove $\mathcal{U}^{+i}$ from our current partition and instead add $\mathcal{U}^{+i,+j}$ and $\mathcal{U}^{+i,-j}$. After solving the respective min-max problems, we have found a partition of $\mathcal{U}$ into three sets. The algorithm terminates and gives a heuristic solution to Problem (M^3) by using an optimal min-max solution on each set of the partition.

Note that if a set $\mathcal{U} \in \mathcal{L}$ maximizing $\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \boldsymbol{c}^\top \boldsymbol{x}$ (see Step 4 of Algorithm 4) is already completely fixed, it consists of only a single cost scenario and cannot be partitioned further. In this case, the heuristic has in fact determined an optimal solution to Problem (M^3), as the objective value has reached the lower bound (MMLB). Algorithm 4 still carries on so that K solutions are found in total.

5. Computational experiments

5.1. Setup

In this section we present the results of computational experiments for the minimization variant of the knapsack problem and the shortest path problem. We show results for the exact row-and-column generation method presented in Algorithm 2 (RCG), the lower bound (MMLB) and both heuristic algorithms (Heur1 and Heur2), presented in Section 4. To analyze the quality of the heuristic solutions we compare the results to the classical min-max solution (MM) and to the solution of a heuristic (HeurPS) already presented in [EKBC19]. The latter heuristic calculates K random Pareto-scenarios of the uncertainty set $\mathcal{U}^\Gamma$ and returns the optimal deterministic solution for each of the scenarios. Note that HeurPS is a more general heuristic that does not exploit the special structure of the budgeted uncertainty set.

All algorithms were implemented in C++. All objective values $cost(\mathbf{x})$ were calculated by solving the IP formulation (5). The lower bound (MMLB) was calculated by using the row-generation procedure presented in Section 3. The min-max problems appearing in the two heuristics are solved via dualized reformulations (e.g., (8) for the first heuristic) rather than solving the polynomial number of deterministic problems, as the dualized MILP appeared to be faster than the latter approach on our instances. All occurring IP and LP formulations as well as the master- and the slave-problem in Algorithm 2, were implemented in CPLEX 12.8. For the minimum knapsack problem and the shortest path problem we used the classical IP formulations. To avoid symmetric solutions with the same objective value we added the symmetry-breaking constraints

$$\sum_{i=1}^{n} i x_i^{(j)} + 1 \leq \sum_{i=1}^{n} i x_i^{(j+1)} \quad j \in [K-1]$$

to the master-problem (Master).

The initial ordering of the indices $i_1, \ldots, i_n$ for Heuristic 1 was selected by sorting the deviations in non-decreasing order $d_{i_1} \leq \ldots \leq d_{i_n}$. This choice was motivated by preliminary tests on random instances.

The Pareto-scenarios for the heuristic presented in [EKBC19] were calculated by drawing K random vectors $\boldsymbol{\lambda}_k$ from a uniform distribution and selecting the Pareto-scenarios

$$\boldsymbol{c}^k := \arg\max_{\boldsymbol{c} \in \mathcal{U}^\Gamma} \boldsymbol{\lambda}_k^\top \boldsymbol{c}.$$

We set a timelimit of 3600 seconds for each of the algorithms. All computations were calculated on a cluster of 64-bit Intel(R) Xeon(R) CPU E5-2603 processors running at 1.60 GHz with 15MB cache. Each algorithm was restricted to one thread.

We consider min-knapsack and shortest path problems. The min-knapsack problem can be written as

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

Our random instances were generated as in [CGKP19]. For each dimension n the costs c_i and the weights w_i were drawn from a uniform distribution on $\{1, \ldots, 100\}$. The knapsack capacity W was set to 35% of the sum of all weights. For each knapsack instance we generated a budgeted uncertainty set with mean vector $\hat{\boldsymbol{c}} = \boldsymbol{c}$ and a random deviation vector $\boldsymbol{d}$ where each d_i is drawn uniformly in $\{1, \ldots, c_i\}$. For each dimension $n \in \{100, 200, 300, 400\}$ we generate 10 random instances and for each instance we vary the parameters $\Gamma \in \{3, 6\}$ and $K \in \{10, 20, 30\}$.

Our shortest path computations were performed on the instances generated in [HKW15]. The authors create graphs with $20, 25, \ldots, 50$ nodes, corresponding to points in the Euclidean plane with random coordinates in $[0, 10]$. They choose a budgeted uncertainty set where $\hat{c}_{ij}$ is set to the Euclidean distance of node i to node j and the deviations are set to $d_{ij} = \frac{c_{ij}}{2}$. The parameter Γ is chosen from $\{3, 6\}$. For each dimension n we tested all 100 instances generated in [HKW15] and for each instance we vary the parameters $\Gamma \in \{3, 6\}$ and $K \in \{10, 20, 30\}$.

5.2. Results on knapsack problems

The results regarding the RCG and the MMLB are shown in Table 2. Each row shows the average over all 10 knapsack instances of the following values (rounded down to one decimal place): the number of items n; the parameter Γ; the number of calculated solutions K; the percental gap (Gap) between the MMLB and the best LB calculated by the RCG during the

timelimit; the total calculation time t in seconds of the MMLB (or RCG); the number of terminated calculations $\#solved$ of the MMLB (or RCG) during the timelimit; the number of iterations $\#iter$ performed by the MMLB (or RCG); the percental optimality gap (Opt-Gap) of the best LB and UB calculated by the RCG during timelimit. Recall that MMLB does not depend on the value of K.

				MMLB			RCG			
n	Γ	K	Gap (%)	t	$\#solved$	$\#iter$	t	$\#solved$	$\#iter$	Opt-Gap (%)
100	3	10	8.0	0.4	10	14.0	3600.0	0	2.0	16.9
100	3	20	8.0	0.4	10	14.0	3600.0	0	2.0	16.9
100	3	30	8.0	0.4	10	14.0	3600.0	0	2.0	16.9
100	6	10	13.4	1.9	10	33.6	3600.0	0	2.1	27.7
100	6	20	13.4	1.9	10	33.6	3600.0	0	2.0	27.8
100	6	30	13.4	1.9	10	33.6	3600.0	0	2.0	27.8
200	3	-	-	3.1	10	37.6				
200	6	-	-	98.2	10	121.9				
300	3	-	-	4.1	10	53.8				
300	6	-	-	292.5	10	197.4				
400	3	-	-	5.8	10	47.0				
400	6	-	-	1639.6	9	247.1				

Table 2: Results of MMLB and RCG for the knapsack problem.

Even for a dimension of $n = 100$ the RCG hit the timelimit of 1 hour in every instance. Furthermore the lower bound given by the MMLB is at least 8% better than the lower bound of the RCG after 1 hour. The optimality gap of the RCG after 1 hour is still at least 17% and sometimes even 27%. Due to this observation and the time consuming calculations of the RCG we did not test the RCG for larger instances. The bounds found using MMLB are stronger. For nearly all configurations we could calculate the bound for all instances during the timelimit. The total calculation time is very small for most of the instances. Interestingly the calculation time increases significantly for the larger uncertainty sets with $\Gamma = 6$ while for $\Gamma = 3$ all instances could be solved in seconds. This is mostly due to the larger number of iterations performed by the MMLB for $\Gamma = 6$.

The results for all three heuristics and the min-max solution are shown in Table 3. Each row shows the average over all 10 knapsack instances of the following values (rounded down to one decimal place): the number of items n; the parameter Γ; the number of calculated solutions K; the percental gap between the MMLB and the solution of Heur1 (or Heur2/HeurPS/MM). We do not record any calculation times here since all procedures return a solution in a few seconds for all instances. The percental gap is always compared to the MMLB since as Table 2 indicates, in nearly all instances this lower bound is tighter than the one provided by the RCG.

n	Γ	K	Heur1	Heur2	HeurPS	MM	n	Γ	K	Heur1	Heur2	HeurPS	MM
100	3	10	2.5	1.3	7.3	5.4	300	3	10	1.5	1.0	3.1	2.7
100	3	20	2.2	0.8	6.5	5.4	300	3	20	1.4	0.7	3.1	2.7
100	3	30	2.5	0.5	6.1	5.4	300	3	30	1.3	0.6	3.1	2.7
100	6	10	3.7	3.2	9.2	7.0	300	6	10	2.7	2.2	5.1	4.2
100	6	20	4.0	1.9	8.8	7.0	300	6	20	2.5	1.8	5.1	4.2
100	6	30	3.6	1.8	8.4	7.0	300	6	30	2.4	1.5	5.1	4.2
200	3	10	1.9	1.0	4.6	3.4	400	3	10	1.0	0.7	2.3	1.9
200	3	20	1.6	0.7	4.6	3.4	400	3	20	0.9	0.5	2.3	1.9
200	3	30	1.7	0.5	4.6	3.4	400	3	30	0.7	0.4	2.3	1.9
200	6	10	3.4	2.7	7.0	5.2	400	6	10	2.1	1.6	3.7	3.0
200	6	20	3.2	2.2	7.0	5.2	400	6	20	1.7	1.4	3.7	3.0
200	6	30	3.1	1.9	7.0	5.2	400	6	30	1.6	1.2	3.7	3.0

Table 3: Percental Gaps of Heur1, Heur2, HeurPS and MM to the lower bound MMLB for the knapsack problem.

Heur2 outperforms the other heuristics for all configurations. The gap to the lower bound is always smaller than 3.2%. The gaps of Heur1 are also very small, at most 3.7%, but always larger than the gaps of Heur2. The gaps of HeurPS are the largest in most of the instances, even larger than the gaps of the min-max solution.

In Figure 3 we show a line plot of the same average gaps as in Table 3 over 10 instances with $n = 150$ and $\Gamma = 6$ for all $K \in \{1, \ldots, n\}$. Heur2 shows the best performance. Heur1 returns solutions which are significantly better than the min-max solutions as well. Unfortunately due to the number

of items $t := \lfloor \frac{n}{K} \rfloor$ considered in each of the $K-1$ subsets in the partition constructed in Heur1, the size of the last set in the partition varies depending on K. This explains the fluctuating gaps of Heur1. The gaps of Heur2 seem to be much more stable, as the algorithm guarantees an improving objective value with increasing K.

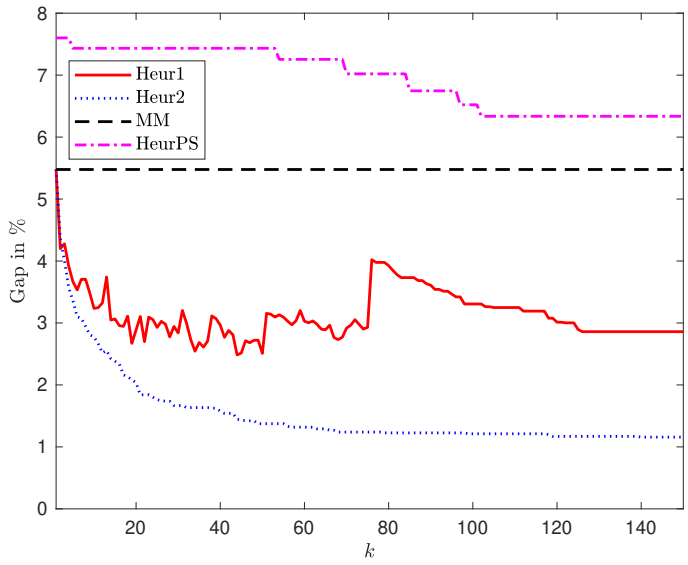

Figure 3: Average percental gaps between objective value of the heuristic solutions and the MMLB over 10 instances with $n = 150$ and $\Gamma = 6$.

5.3. Results on shortest path problems

In this section we consider the classical shortest path problem. The results regarding the RCG and the MMLB are shown in Table 4. The results for all three heuristics and the min-max solution are shown in Table 5. Each row shows the average over all 100 shortest path instances.

Even for a dimension of $n = 131$ the RCG hit the timelimit of 1 hour in every instance. For most configurations it could solve at most 25 of 100 instances during the timelimit. Furthermore the lower bound given by the

				MMLB			RCG			
n	Γ	K	Gap (%)	t	$\#solved$	$\#iter$	t	$\#solved$	$\#iter$	Opt-Gap (%)
57	3	10	5.4	0.0	100	7.3	3279.9	14	4.8	9.7
57	3	20	5.4	0.0	100	7.3	3428.3	10	6.0	9.8
57	3	30	5.8	0.0	100	7.3	3360.7	14	5.0	11.0
57	6	10	8.4	0.1	100	12.0	3453.1	9	5.0	17.1
57	6	20	8.2	0.1	100	12.0	3535.7	7	5.4	16.2
57	6	30	9.0	0.1	100	12.0	3572.3	6	5.2	17.7
90	3	10	6.4	0.1	100	8.8	3507.4	4	4.3	13.5
90	3	20	6.8	0.1	100	8.8	3566.8	2	3.6	15.4
90	3	30	8.5	0.1	100	8.8	3567.4	1	3.1	18.8
90	6	10	10.4	0.2	100	15.0	3568.9	1	3.9	24.9
90	6	20	11.5	0.2	100	15.0	3600.0	0	3.5	26.7
90	6	30	12.6	0.2	100	15.0	3600.0	0	3.0	30.0
131	3	10	6.6	0.1	100	10.1	3600.0	0	4.1	16.6
131	3	20	7.4	0.1	100	10.1	3600.0	0	3.6	18.5
131	3	30	8.8	0.1	100	10.1	3600.0	0	3.0	22.5
131	6	10	11.4	0.3	100	18.3	3600.0	0	3.6	30.9
131	6	20	12.5	0.3	100	18.3	3600.0	0	3.3	32.1
131	6	30	13.5	0.3	100	18.3	3600.0	0	2.9	34.2
179	3	-	-	0.1	100	12.0				
179	6	-	-	0.6	100	22.9				
234	3	-	-	0.2	100	11.8				
234	6	-	-	0.6	100	23.7				
297	3	-	-	0.2	100	12.5				
297	6	-	-	0.9	100	28.2				
368	3	-	-	0.3	100	13.7				
368	6	-	-	1.2	100	32.3				

Table 4: Results of MMLB and RCG for the shortest path problem.

MMLB is larger than the lower bound of the RCG after 1 hour. For small instances it is at least 5% better while for the larger instances the gap increases up to 12% for some instances. The optimality gap of the RCG after 1 hour is still at least 9%, for larger instances even around 30%. Due to this observation and the time consuming calculations of the RCG we did not test the RCG for larger instances as the bounds provided by MMLB are tighter and the hardest of them could be computed in at most 1.2 seconds. This is due to the very small number of iterations and the small computational effort of the shortest path problem in its deterministic version.

In Table 5 we show the results for all three heuristics and the min-max

n	Γ	K	Heur1	Heur2	HeurPS	MM
57	3	10	5.1	1.7	8.1	15.3
57	3	20	5.6	0.5	6.1	15.3
57	3	30	7.4	0.3	5.5	15.3
57	6	10	8.2	3.9	8.7	17.4
57	6	20	7.9	2.1	6.5	17.4
57	6	30	9.1	1.7	5.8	17.4
90	3	10	5.1	2.3	11.8	18.0
90	3	20	5.1	0.9	9.5	18.0
90	3	30	4.6	0.5	8.6	18.0
90	6	10	9.5	5.7	14.1	22.6
90	6	20	8.6	3.3	11.2	22.6
90	6	30	8.0	2.4	10.2	22.6
131	3	10	6.4	3.4	15.7	19.9
131	3	20	5.8	1.7	13.6	19.9
131	3	30	5.7	0.9	13.0	19.9
131	6	10	10.9	7.5	20.4	26.1
131	6	20	8.9	4.8	18.0	26.1
131	6	30	9.4	3.6	16.5	26.1
179	3	10	7.3	5.2	20.2	22.4
179	3	20	7.1	3.2	17.7	22.4
179	3	30	7.9	2.2	15.8	22.4
179	6	10	11.4	8.9	23.1	29.6
179	6	20	10.5	5.8	20.2	29.6
179	6	30	10.7	4.7	18.7	29.6
234	3	10	7.3	5.7	22.5	22.7
234	3	20	6.8	3.3	20.1	22.7
234	3	30	6.9	2.1	19.1	22.7
234	6	10	11.7	9.8	27.5	31.0
234	6	20	11.0	6.5	25.0	31.0
234	6	30	10.0	5.1	23.5	31.0
297	3	10	7.7	6.5	24.1	22.9
297	3	20	7.3	4.5	20.9	22.9
297	3	30	7.7	3.1	20.0	22.9
297	6	10	12.3	10.4	29.9	32.3
297	6	20	10.9	7.5	28.1	32.3
297	6	30	10.6	5.9	26.8	32.3
368	3	10	7.5	7.3	25.5	23.3
368	3	20	6.9	5.2	23.7	23.3
368	3	30	6.7	3.5	22.8	23.3
368	6	10	13.3	12.5	32.5	33.7
368	6	20	11.1	9.1	30.9	33.7
368	6	30	10.8	7.5	29.7	33.7

Table 5: Percental Gaps of Heur1, Heur2, HeurPS and MM to the lower bound MMLB for the shortest path problem.

solution. Heur2 outperforms all other heuristics for all configurations. Compared to the knapsack problem the gaps are slightly larger for higher dimensions but are never larger than 12%. The gaps of Heur1 are also larger than for the knapsack problem, at most 13.3%, but always larger than the gaps of Heur2. In contrast to the knapsack problem here the min-max solution provides the worst gaps in nearly all instances. The gaps of HeurPS are slightly better but can also increase up to 30% for larger instances. To summarize Heur1 and Heur2 seem to be a good choice to solve Problem (M^3) for the shortest path problem.

In Figure 4 we show a line plot of the same average gaps as in Table 5 over 100 instances with $n = 179$ and $\Gamma = 6$ for all $K \in \{1, \ldots, n\}$. Again, Heur2 outperforms the other heuristics. Heur1 returns solutions which are

significantly better than the min-max solutions as well. In contrast to the knapsack problem the HeurPS performs better than the min-max solution here.

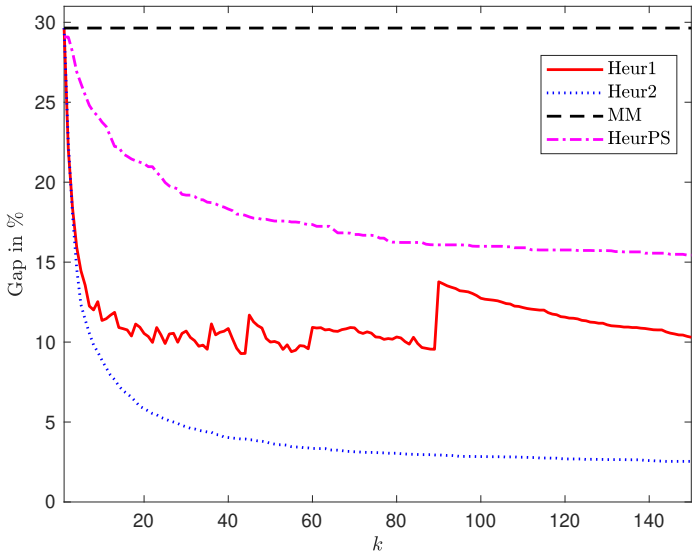

Figure 4: Average percental gaps between objective value of the heuristic solutions and the MMLB over 100 instances with $n = 179$ and $\Gamma = 6$.

6. Conclusion

We considered the min-max-min problem in robust combinatorial optimization, where it is possible to prepare K solutions beforehand. Once the uncertain costs are revealed, one then chooses the best of the prepared solutions for this scenario. For the first time, the min-max-min setting is considered in combination with discrete budgeted uncertainty.

Our complexity analysis reveals that most combinatorial problems become $\mathcal{NP}$-hard in this setting, and even inapproximable. Furthermore, even evaluating the objective value of a K-tuple of solutions is already $\mathcal{NP}$-hard,

making it unlikely that a compact problem formulation exists. We thus present a row-and-column generation approach to find exact solutions. As this approach fails for larger problem instances, we also develop two heuristic algorithms that run in polynomial time. Computational experiments indicate that these heuristics scale well with the problem size, leading to solutions in seconds that leave a gap of a few percent for large instances when compared to a simple lower bound.

References

[ABV09] 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.

[ANSdG12] Sibel A Alumur, Stefan Nickel, and Francisco Saldanha-da Gama. Hub location under uncertainty. *Transportation Research Part B: Methodological*, 46(4):529–543, 2012.

[BDPZ18] Dimitris Bertsimas, Jack Dunn, Colin Pawlowski, and Ying Daisy Zhuo. Robust classification. *INFORMS Journal on Optimization*, 1(1):2–34, 2018.

[BK17] Christoph Buchheim and Jannis Kurtz. Min–max–min robust combinatorial optimization. *Mathematical Programming*, 163(1):1–23, 2017.

[BK18a] Christoph Buchheim and Jannis Kurtz. Complexity of min–max–min robustness for combinatorial optimization under discrete uncertainty. *Discrete Optimization*, 28:1–15, 2018.

[BK18b] Christoph Buchheim and Jannis Kurtz. Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO Journal on Computational Optimization*, 6(3):211–238, 2018.

[BS03] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Math. Program.*, 98(1-3):49–71, 2003.

[CGKP19] André B. Chassein, Marc Goerigk, Jannis Kurtz, and Michael Poss. Faster algorithms for min-max-min robustness for combinatorial problems with budgeted uncertainty. *European Journal of Operational Research*, 279(2):308–319, 2019.

[CGKZ18] André Chassein, Marc Goerigk, Adam Kasperski, and Paweł Zieliński. On recoverable and two-stage robust selection problems with budgeted uncertainty. *European Journal of Operational Research*, 265(2):423–436, 2018.

[Cha17] André Chassein. Having a plan B for robust optimization. Technical report, Technische Universität Kaiserslautern, 2017.

[CTC07] 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.

[EKBC19] Lars Eufinger, Jannis Kurtz, Christoph Buchheim, and Uwe Clausen. A robust approach to the capacitated vehicle routing problem with uncertain costs. *INFORMS Journal on Optimization*, 2019. To appear.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[HKW15] Grani A Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891, 2015.

[HS76] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 1976.

[IPS82] Alon Itai, Yehoshua Perl, and Yossi Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3):277–286, 1982.

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, Berlin, 2004.

[KZ16] 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.

[LMSL90] Chung-Lun Li, S Thomas McCormick, and David Simchi-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete Appl. Math.*, 26(1):105–115, 1990.

[LPdB+13] 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.

[Pos18] Michael Poss. Robust combinatorial optimization with knapsack uncertainty. *Discrete Optimization*, 27:88–102, 2018.

[SGW17] Anirudh Subramanyam, Chrysanthos E Gounaris, and Wolfram Wiesemann. K-adaptability in two-stage mixed-integer robust optimization. *arXiv preprint arXiv:1706.07097*, 2017.

[ZZ13] Bo Zeng and Long Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

Appendix A. Additional proofs

Theorem 11. *Problem* (M^3) *for the spanning tree problem is weakly $\mathcal{NP}$-hard, even if $K = 2$. It is strongly $\mathcal{NP}$-hard if K is part of the input.*

Proof. Assume first that $\Gamma = 1$ and $K = 2$. We reduce the 2-partition problem to (M^3) for the spanning tree problem. Given an instance of the 2-partition problem i.e. $a_i \in \mathbb{N}$ for each $i \in N = [n]$ we define the graph $G = (V, E)$ as follows: Let $V = \{v_1, \ldots, v_{n+1}, w_1, \ldots, w_{n+1}\}$ and $E = \{e_1, \ldots, e_n, f_1, \ldots, f_n, g_1, \ldots, g_{n+1}\}$ where $e_i = \{v_i, v_{i+1}\}$, $f_i = \{w_i, w_{i+1}\}$ and $g_i = \{v_i, w_i\}$. Assume e_i and f_i have nominal costs a_i and a deviation of M where $M = \sum_{i\in[n]} a_i$. All edges g_i have costs and deviation 0. Note that all optimal spanning trees of the latter graph must use all edges g_i and for each $i \in N$ exactly one of the edges e_i or f_i. Now any optimal solution of (M^3) for $K = 2$ contains two trees which are disjoint on the e-edges and on the f-edges since otherwise the deviation on a common edge could be set to M. Thus we can find a solution $I \subseteq N$ for the 2-partition problem if and only if the optimal value of (M^3) is $\frac{1}{2}M$.

The proof extends to the case when K is part of the input by setting $\Gamma = K - 1$ and constructing a graph $G = (V, E)$ with nodes

$$V = \left\{v_1^1, \ldots, v_{n+1}^1, \ldots, v_1^K, \ldots, v_{n+1}^K\right\}$$

and edges

$$E = \left\{e_1^1, \ldots, e_n^1, \ldots, e_1^K, \ldots, e_n^K, g_1^1, \ldots, g_{n+1}^1, \ldots, g_1^{K-1}, \ldots, g_{n+1}^{K-1}\right\}$$

where $e_i^j = \left\{v_i^j, v_{i+1}^j\right\}$ and $g_i^j = \left\{v_i^j, v_i^{j+1}\right\}$. Again all edges e_i^j have nominal costs a_i and a deviation of M where $M = \sum_{i\in[n]} a_i$. All edges g_i^j have costs and deviation 0. A similar reasoning as above shows that one can decide if a partition into K sets having the same costs exists if and only if the

corresponding min-max-min problem has an optimal value equal to $\frac{1}{K}M$. The problem generalizes the 3-partition problem, which is $\mathcal{NP}$-hard in the strong sense, which proves the result. □

Theorem 12. *Problem* (M^3) *for the assignment problem is weakly* $\mathcal{NP}$*-hard, even if* $K = 2$*. It is strongly* $\mathcal{NP}$*-hard if* K *is part of the input.*

Proof. Assume first that $\Gamma = 1$ and $K = 2$. We reduce the 2-partition problem to (M^3) for the assignment problem. Given an instance of the 2-partition problem i.e. $a_i \in \mathbb{N}$ for each $i \in N = [n]$ we want to know if there exists a subset $I \subseteq N$ with $|I| = |N \setminus I|$ such that $\sum_{i \in I} a_i = \sum_{i \in N \setminus I} a_i$. We consider a graph $G = (V, E)$ with nodes $V = \{v_1, \ldots, v_n, w_1, \ldots, w_n\}$ and edges $E = \{\{v_i, w_j\} : i, j = 1, \ldots, n\}$. The edges $\{v_i, w_i\}$ have nominal costs $-a_i$ and deviation $M = \sum_{i \in [n]} a_i$ for each $i = 1, \ldots, n$. All other edges have costs and deviation 0. By the choice of M the two solutions in an optimal solution of (M^3) are disjoint and each edge $\{v_i, w_i\}$ is used by at least one of the two solutions. Thus we can find a solution $I \subseteq N$ for the 2-partition problem with $|I| = |N \setminus I|$ if and only if the optimal value of (M^3) is $-\frac{1}{2}M$.

The proof extends to the case when K is part of the input by the same construction and $\Gamma = K - 1$. A similar reasoning as above shows that one can decide if a partition into K sets having the same costs exists if and only if the corresponding min-max-min problem has an optimal value equal to $\frac{1}{K}M$. The problem generalizes the 3-partition problem, which is $\mathcal{NP}$-hard in the strong sense, which proves the result.

□

Appendix B. Dynamic programming for the knapsack problem with any fixed K and Γ

Consider first $\Gamma \geq 2$ and $K = 2$. As before, the algorithm enumerates labels $\boldsymbol{s}$ and chooses the best of them by computing their costs. As $K =$

2, only two solutions are being built, and steps 6–12 follow the same idea as before with one difference: computing the cost (2) requires the worst Γ deviations for each partial solution. Therefore, every state $s \in \mathcal{S}$ is now described by the $(3\Gamma + 4)$-tuple $\boldsymbol{s} = (w^{(1)}, w^{(2)}, c^{(1)}, c^{(2)}, \mathbf{i}^{(1)}, \mathbf{i}^{(2)}, \mathbf{i}^{(1,2)})$ where $\mathbf{i}^{(1)}$, $\mathbf{i}^{(2)}$, and $\mathbf{i}^{(1,2)}$ are Γ-tuples recording the indices of the largest elements. Equation (2) becomes

$$cost(\boldsymbol{s}) = \max_{\substack{S \subseteq [n] \\ |S| \leq \Gamma}} \left\{ \min \left(c^{(1)} + \sum_{i \in S \cap (\mathbf{i}^{(1)} \cup \mathbf{i}^{(1,2)})} d_i, c^{(2)} + \sum_{i \in S \cap (\mathbf{i}^{(2)} \cup \mathbf{i}^{(1,2)})} d_i \right) \right\}. \tag{B.1}$$

For $K \geq 3$, we are now constructing K solutions, so steps 6–12 should be adapted accordingly. In addition, computing the cost for $K \geq 3$ also requires the Γ worst deviations for each subset $\mathbf{j} \subseteq [K]$ that contains at most Γ elements. We obtain states described by

$$\boldsymbol{s} = (w^{(1)}, \ldots, w^{(K)}, c^{(1)}, \ldots, c^{(K)}, \mathbf{i}^{(\mathbf{j}_1)}, \ldots, \mathbf{i}^{(\mathbf{j}_{\mathfrak{K}})}),$$

where

$$\mathfrak{K} = \sum_{\gamma=1}^{\min(K,\Gamma)} \binom{K}{\gamma},$$

which is constant when K and Γ are constant. The resulting set $\mathcal{S}$ contains $O(n^{K+\Gamma\mathfrak{K}}\bar{c}^K)$ many states. The cost function (B.1) can be extended similarly.