Min-max-min robust combinatorial optimization with few recourse solutions

Ayşe Nur Arslan, Michael Poss, Marco Silva

Min-max-min robust combinatorial optimization with few recourse solutions

Ayşe Nur Arslan
IRMAR, INSA de Rennes, Rennes, France, ayse-nur.arslan@insa-rennes.fr,

Michael Poss
LIRMM, University of Montpellier, CNRS, France, michael.poss@lirmm.fr,

Marco Silva
CEGI, INESCTEC, Porto, Portugal, marco.c.silva@inesctec.pt,

In this paper, we consider a variant of adaptive robust combinatorial optimization problems with cost uncertainty where the decision maker can prepare K solutions and choose the best among them upon knowledge of the true cost vectors. We propose a new exact algorithm for solving these problems when the feasible set of the nominal optimization problem does not contain too many good solutions. Our algorithm enumerates these good solutions, generates dynamically a set of scenarios from the uncertainty set, and assigns the solutions to the generated scenarios using a vertex p-center formulation, solved by a binary search algorithm. Our numerical results on adaptive shortest path and knapsack with conflicts problems show that our algorithm compares favorably with the methods proposed in the literature. We additionally propose a heuristic extension of our method to handle problems where it is prohibitive to enumerate all good solutions. This heuristic is shown to provide good solutions within a reasonable solution time limit on the adaptive knapsack with conflicts problem.

Key words: robust optimization, combinatorial optimization, vertex p-center, scenario generation
History:

1. Introduction

Robust optimization is an approach to handling uncertainty in optimization where the probability distributions are replaced with uncertainty sets. In robust optimization, constraints are imposed for all realizations whereas the objective function is evaluated for the worst-case realization within the uncertainty set. As such, in applications where the effects of uncertainty can be catastrophic, robust optimization presents itself as a viable

modeling approach. Further, static robust optimization models with polyhedral or convex uncertainty sets lead to deterministic equivalent formulations that are often in the same complexity class as their deterministic counterparts. For these reasons, robust optimization has enjoyed and continues to enjoy a growing attention from the research community. Advances in static robust optimization are presented in Ben-Tal et al. (2009), Bertsimas et al. (2011), Buchheim and Kurtz (2018b), Gabrel et al. (2014).

The static robust optimization framework was extended to combinatorial optimization problems in Kouvelis and Yu (2013). Given the feasibility set of a combinatorial optimization problem, the framework considers that the cost vector can take any value in a given uncertainty set and seeks a solution that is optimal under the worst-case realization over this set. Assuming that the uncertainty set is a polytope described by a constant number of inequalities (such as the budget uncertainty set Bertsimas and Sim (2003)), the static robust combinatorial optimization problem is not much harder than its deterministic counterpart, since solving it amounts to solve a polynomial number of problems for different cost vectors Bertsimas and Sim (2003), Lee and Kwon (2014), Poss (2018). However, when the uncertainty sets is defined by a non-constant number of inequalities, or is simply defined by a non-constant list of possible cost vectors, the robust counterpart is typically harder than its deterministic variant Aissi et al. (2009), Kouvelis and Yu (2013).

Static robust combinatorial optimization problems are often considered too conservative since no recourse action can be taken to undermine the effect of uncertainty. A natural extension, coined *the min-max-min robust combinatorial optimization problem*, has therefore been proposed by Buchheim and Kurtz (2017, 2018a) allowing the decision maker to choose K solutions before the realization of uncertainty, and then, to select the best among them when the uncertain parameters materialize, thus mitigating the adverse effects. The min-max-min robust combinatorial optimization problem models the situation where the decision maker can prepare the ground for K solutions (e.g. training drivers, repairing roads, configure routing protocols) and choose the best among them upon full knowledge of the uncertain parameters. For instance, if the feasible set contains paths in a given graph, the problem seeks to prepare K different routes that can be used to evacuate citizens or transport relief supplies in case of a hazardous event Hanasusanto et al. (2015). Another relevant example is related to parcel delivery companies, which would be unwilling to reschedule their delivery tours from scratch everyday. As an alternative, drivers are

trained only for a small set of route plans that are to be executed in case of road closures and heavy deviations Eufinger et al. (2019), Subramanyam et al. (2019).

While several studies (e.g., Eufinger et al. (2019), Hanasusanto et al. (2015), Subramanyam et al. (2019)) have illustrated the practical relevance of the min-max-min robust combinatorial optimization problem, exact solution algorithms have stayed behind, and the problem has remained extremely difficult to solve, even for small instances. Two general algorithms have been proposed: Hanasusanto et al. (2015) reformulates the problem through linear programming duality resulting in a monolithic formulation involving bilinear terms that are linearized using the McCormick linearization, and Subramanyam et al. (2019) introduces an ad-hoc branch-and-bound algorithm based on generating a relevant subset of uncertainty realizations and enumerating over their assignment to the K solutions. Unfortunately, these two approaches can hardly solve the shortest path instances proposed by Hanasusanto et al. (2015) as soon as the graph contains more than 25 nodes. A third paper has had more success with these instances, solving all of them to optimality (the largest having 50 nodes) in the special case $K \in \{2, 3\}$ Chassein et al. (2019). Yet this latter approach requires the uncertainty set to be the budgeted uncertainty set from Bertsimas and Sim (2003) and does not scale up with K. The theoretical complexity of the problem with budgeted uncertainty has also been investigated in Goerigk et al. (2020) (for the discrete variant) and Chassein and Goerigk (2020) (for its continuous variant). In addition, Goerigk et al. (2020) proposes heuristic algorithms with no performance guarantee. It should be noted that Hanasusanto et al. (2015), Subramanyam et al. (2019) address, in fact, the more general K-adaptability paradigm, in which first-stage decisions can be taken in addition to the K different second-stage decisions. The paradigm of K-adaptability is strongly related to two-stage robust optimization with discrete recourse, a topic having witnessed a significant amount of work in the past few years (e.g. Arslan and Detienne (2020), Bertsimas and Dunning (2016), Bertsimas and Georghiou (2018)).

Our contribution The purpose of this work is to propose a novel algorithm for solving the min-max-min robust combinatorial optimization problem. Borrowing an idea from Chassein et al. (2019), our algorithm enumerates *good* feasible solutions. However, unlike Chassein et al. (2019) that relies on a purely enumerative scheme tailored for the budgeted uncertainty set from Bertsimas and Sim (2003), the approach presented in this paper models the problem with a generic uncertainty polytope as a variant of the p-center problem,

assigning a *relevant* subset of scenarios to at most K different solutions. We solve the resulting problem by combining a row-and-column generation algorithm, binary search, and dominance rules. In doing so, we are able to optimally solve many open instances from Hanasusanto et al. (2015), in particular, when K grows above 3. Our approach is not impacted by the complexity of the nominal counterpart of the problem, unlike Hanasusanto et al. (2015) and Subramanyam et al. (2019). This property is illustrated numerically on the min-max-min counterpart of the knapsack problem with conflicts, for which our algorithm compares even more favorably against the approaches from Hanasusanto et al. (2015) and Subramanyam et al. (2019). We additionally propose a heuristic variant of our algorithm based on a partial enumeration of good feasible solutions that can be used when enumerating all good solutions is computationally prohibitive. The latter is compared numerically to the heuristic variant of the branch-and-bound algorithm from Subramanyam et al. (2019) on the the knapsack problem with conflicts. The results indicate that our heuristic provides slightly better quality solutions for the larger values of k. The source code of the algorithms presented in this paper is available here.

Structure of the paper The rest of the paper is organized as follows. In Section 2, we formally define the min-max-min combinatorial optimization problem, and describe the details of our proposed solution scheme. In Section 3, we present the numerical results comparing our solution scheme to the well established methodologies from the K-adaptability literature on the shortest path problem and the knapsack problem with conflicts. We conclude the paper in Section 4.

Notations Troughout the paper, we use the short-hand notations $[n] = \{1, \ldots, n\}$ for any positive integer n, and $\mathbf{x} = (x^1, \ldots, x^K)$ where x^i for $i \in [K]$ is a feasible solution of the set X.

2. Methodological development

In this section, we formally define the min-max-min robust combinatorial optimization problem. We then present the different components of the solution scheme we propose.

2.1. Problem definition

Given the feasibility set $X \subseteq \{0,1\}^n$ of a combinatorial optimization problem, the robust combinatorial optimization framework considers that the cost vector can take any value in

the set $\Xi \subseteq \mathbb{R}^n$ and seeks a solution $x \in X$ that is optimal under the worst-case realization over Ξ, thus solving

$$\min_{x \in X} \max_{\xi \in \Xi} \ \xi^\top x. \tag{1}$$

A natural extension of (1) is to allow the decision maker to choose K solutions $x^1, \ldots, x^K$ in X, and then, select the best among them when the uncertain parameters materialize. We formally state the min-max-min robust combinatorial optimization problem as

$$v := \min_{\mathbf{x} \in X^K} \max_{\xi \in \Xi} \min_{k \in [K]} \ \xi^\top x^k. \tag{2}$$

We illustrate problem (2) with an example on the shortest path problem presented in Figure 1, where Ξ contains two scenarios (indicated as a tuple on each arc in Figure 1) and M is a large value representing a hazardous event that would cause road closures. Solving the classical robust problem leads to a solution of cost $M+1$. In contrast, setting $K=2$ we obtain $x^1 = \{(1,2),(2,4)\}$, $x^2 = \{(1,3),(3,4)\}$, so under any scenario realization, there remains an alternative route of cost 2.

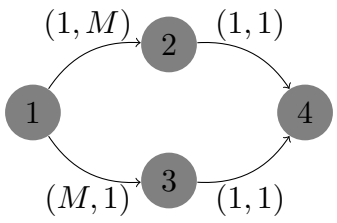

Figure 1 **Shortest path example with $X = \{\{(1,2),(2,4)\},\{(1,3),(3,4)\}\}$. Ξ contains two scenarios that are indicated as a tuple on each arc.**

2.2. Scenario generation

The algorithm developed in this paper lies in generating a subset of scenarios dynamically, following recent line of research addressing difficult robust optimization problems, e.g. Agra et al. (2016), Ayoub and Poss (2016), Bertsimas et al. (2016), Fischetti and Monaci (2012), Pessoa and Poss (2015), Subramanyam et al. (2019), Zeng and Zhao (2013) among others. To this end, let $\tilde{\Xi} \subseteq \Xi$ be a subset of scenarios of finite cardinality. We consider a relaxation of (2) where Ξ is replaced by the subset $\tilde{\Xi} \subseteq \Xi$, namely

$$v(\tilde{\Xi}) := \min_{\mathbf{x} \in X^K} \max_{\xi \in \tilde{\Xi}} \min_{k \in [K]} \ \xi^\top x^k. \tag{3}$$

Let $\bar{\mathbf{x}}$ be an optimal solution to (3). While $\bar{\mathbf{x}}$ may be suboptimal for (2), the value of (3) provides a lower bound for the optimal value of (2), since (3) is a relaxation of (2). The following proposition formalizes this observation.

PROPOSITION 1. *Let* $\tilde{\Xi} \subseteq \Xi$ *then* $v(\tilde{\Xi}) \leq v$.

Proof. Let $\hat{\mathbf{x}}$ be any optimal solution to (2). We obtain

$$\begin{aligned} v &= \max_{\xi \in \Xi} \min_{k \in [K]} \ \xi^\top \hat{x}^k \\ &\geq \max_{\xi \in \tilde{\Xi}} \min_{k \in [K]} \ \xi^\top \hat{x}^k \\ &\geq \max_{\xi \in \tilde{\Xi}} \min_{k \in [K]} \ \xi^\top \bar{x}^k = v(\tilde{\Xi}), \end{aligned}$$

where the first inequality follows since $\tilde{\Xi} \subseteq \Xi$ and the second follows from the optimality of $\bar{\mathbf{x}}$ in (3). $\square$

We next describe how to augment the subset $\tilde{\Xi}$ with relevant uncertainty realizations. Given an optimal solution $\bar{\mathbf{x}}$ to (3), the optimality of $\bar{\mathbf{x}}$ in (2) can be tested by verifying whether its true objective value is equal to the lower bound $v(\tilde{\Xi})$. In other words, one needs to compute

$$v(\bar{\mathbf{x}}) := \max_{\xi \in \Xi} \min_{k \in [K]} \ \xi^\top \bar{x}^k, \tag{4}$$

which can be linearized through an epigraph reformulation:

$$\begin{aligned} v(\bar{\mathbf{x}}) = \max \quad & \eta \\ \text{s.t.} \quad & \xi \in \Xi \\ & \eta \leq \xi^\top \bar{x}^k, \quad k \in [K]. \end{aligned}$$

The value of 4 provides an upper bound for the optimal value of (2) as it is the true value of a given feasible solution $\bar{\mathbf{x}}$. This observation is formalized in the following proposition.

PROPOSITION 2. *Let* $\bar{\mathbf{x}} \in X^K$ *then* $v \leq v(\bar{\mathbf{x}})$.

Proof. $v(\bar{\mathbf{x}}) = \max_{\xi \in \Xi} \min_{k \in [K]} \ \xi^\top \bar{x}^k \leq \min_{\mathbf{x} \in X^K} \max_{\xi \in \Xi} \min_{k \in [K]} \ \xi^\top x^k = v.$ $\square$

Propositions 1 and 2 immediately imply that if $v(\tilde{\Xi}) = v(\bar{\mathbf{x}})$, then the solution $\bar{\mathbf{x}}$ is an optimal solution of (2). On the other hand, if $v(\tilde{\Xi}) < v$, we have no guarantee regarding

the optimality of $\bar{\mathbf{x}}$ and our algorithm proceeds by adding to $\tilde{\Xi}$ any scenario ξ^* that reaches the maximum in (4), specifically

$$\xi^* \in \arg\max_{\xi\in\Xi} \min_{k\in[K]} \xi^\top \bar{x}^k.$$

The overall scenario generation scheme is provided in Algorithm 1, wherein ϵ is a numerical tolerance factor, and ξ^0 is any scenario from Ξ (typically chosen to be the nominal scenario). A proof of its finite convergence is provided in Proposition 3.

Algorithm 1: Scenario generation.

Initialization: $UB = \min_{x\in X} \max_{\xi\in\Xi} \xi^\top x$, $LB = \min_{x\in X} \min_{\xi\in\Xi} \xi^\top x$, $\tilde{\Xi} = \{\xi^0\}$, $\mathbf{x}^* = null$;
repeat
  Compute $v(\tilde{\Xi})$ and $\bar{\mathbf{x}}$ by solving (3);
  $LB \leftarrow v(\tilde{\Xi})$;
  Compute $v(\bar{\mathbf{x}})$ by solving (4);
  if $v(\bar{\mathbf{x}}) < UB$ **then**
    $UB \leftarrow v(\bar{\mathbf{x}})$;
    $\mathbf{x}^* \leftarrow \bar{\mathbf{x}}$;
  end
  if $\frac{UB-LB}{LB} > \epsilon$ **then**
    $\tilde{\Xi} \leftarrow \tilde{\Xi} \cup \{\xi^*\}$;
    scenario_added = ***true***;
  end
  else *scenario_added* = ***false***;
until *scenario_added* = ***false***;
Return: $\mathbf{x}^*$

PROPOSITION 3. *Algorithm 1 converges in a finite number of iterations independent of the value of ϵ.*

Proof. Let us denote by $\bar{\mathbf{x}}^i$ the i-th optimal solution to (3) computed along the algorithm, corresponding to the set $\tilde{\Xi}^i$, and yielding the upper bound $v(\bar{\mathbf{x}}^i)$ obtained by the uncertain vector $\xi^{*^i} \in \Xi$. Consider $j > i$ such that $\bar{\mathbf{x}}^j = \bar{\mathbf{x}}^i$ and recall that by definition of ξ^{*^i},

$v(\bar{\mathbf{x}}^i) = \min_{k\in[K]} \ (\xi^{*^i})^\top \bar{x}^{k,i}$. Thus, $\xi^{*^i} \in \tilde{\Xi}^j$ implies that $v(\tilde{\Xi}^j) = \max_{\xi\in\tilde{\Xi}^j} \min_{k\in[K]} \xi^\top \bar{x}^{k,j} = \max_{\xi\in\tilde{\Xi}^j} \min_{k\in[K]} \xi^\top \bar{x}^{k,i} \geq \min_{k\in[K]} \ (\xi^{*^i})^\top \bar{x}^{k,i} = v(\bar{\mathbf{x}}^i) = v(\bar{\mathbf{x}}^j)$, proving the optimality of solution $\bar{\mathbf{x}}^j = \bar{\mathbf{x}}^i$. Since the number of feasible solution vectors $\mathbf{x} \in X^K$ is finite the algorithm therefore converges finitely. $\square$

2.3. Vertex p-center formulation

One of the numerical challenges of Algorithm 1 is the efficient solution of problem (3) to optimality. In this section, we detail how problem 3 can be cast as a vertex $p-$center problem and be solved using a binary search algorithm as a result. To this end, let X be the enumerated set $(x_1, \ldots, x_r)$, meaning that the underlying nominal problem contains r different feasible solutions. Notice that r is typically very large so that we will provide techniques to restrict ourselves to a relevant subset of these feasible solutions (see Section 2.4). We also let $\tilde{\Xi}$ be the enumerated set $(\xi_1, \ldots, \xi_m)$ where m denotes the number of scenarios generated so far.

Having the aforementioned two enumerated sets in mind, problem (3) can be reformulated as follows: we must choose at most K feasible solutions such that each scenario is assigned to one of these K solutions and such that the worst-case assignment cost is minimized. Let the binary decision variable z_s, for $s \in [r]$, equal to 1 if and only if feasible solution x_s is selected. Further, for $s \in [r]$ and $j \in [m]$, let the binary decision variable y_{sj} equal to 1 if and only if the feasible solution x_s is assigned to the scenario ξ_j. The problem then formalizes as:

$$\min \quad \omega \tag{5}$$

$$\text{s.t.} \quad \omega \geq \sum_{s\in[r]} \xi_j^\top x_s y_{sj}, \qquad \forall j \in [m] \tag{6}$$

$$\sum_{s\in[r]} y_{sj} = 1, \qquad \forall j \in [m] \tag{7}$$

$$\sum_{s\in[r]} z_s \leq K, \tag{8}$$

$$y_{sj} \leq z_s, \qquad \forall\ s \in [r], j \in [m] \tag{9}$$

$$z \in \{0,1\}^r, y \in \{0,1\}^{r\times m}. \tag{10}$$

Based on this formulation, problem (3) is a vertex p-center problem (introduced in Daskin (2011)) where X contains the possible centers, $\tilde{\Xi}$ contains the clients that need to be served, and $p = K$.

We next consider the decision version of this problem and look for a solution of cost not greater than a given threshold $\rho > 0$. Since we are looking for a solution with a cost not greater than ρ, we can disregard assignments with costs greater than ρ. We model this with the help of the 0/1 coverage matrix cov defined as follows: given a threshold ρ, cov_{sj} equals 1 if and only if $\xi_j^\top x_s \leq \rho$ for $s \in [r]$ and $j \in [m]$. With the above definitions, the optimal value of problem (3) is not greater than ρ if and only if there exists a feasible solution to the following system (see Minieka (1970)):

$$\sum_{s\in[r]} cov_{sj} y_{sj} = 1, \quad \forall j \in [m] \tag{11}$$

$$\sum_{s\in[r]} z_s \leq K, \tag{12}$$

$$y_{sj} \leq z_s, \quad \forall s \in [r],\ j \in [m] \tag{13}$$

$$y \in \{0,1\}^{r\times m}, z \in \{0,1\}^r. \tag{14}$$

We remark that checking the feasibility of the above system is $\mathcal{NP}$-hard, and the best one can do is to enumerate all possible assignments Chalermsook et al. (2017). Herein, we solve the problem by a mixed-integer linear program (MILP), replacing constraints (11) with

$$\sum_{s\in[r]} cov_{sj} y_{sj} \geq 1 - \alpha_j, \quad \forall j \in [m], \tag{15}$$

and minimizing the objective function

$$\sum_{j\in[m]} \alpha_j. \tag{16}$$

If the optimal value of the resulting MILP is zero, then the optimal value of (3) is at most ρ, so that we can reduce the threshold ρ. Otherwise, there is no feasible solution with cost at most ρ, so we must increase the threshold to recover feasibility. This results in a binary search on the optimal value of ρ, described in Algorithm 2 allowing to solve problem (3) to optimality. Similar algorithms have been proposed for solving the vertex p-center problem, e.g. Chen and Chen (2009), Contardo et al. (2019).

The above scheme can be improved by removing dominated solutions and scenarios, where a solution s is said to be *dominated* if there exists $s' \in [r]$ such that $cov_{sj} \leq cov_{s'j}$ for each $j \in [m]$, while a scenario j is said to be *dominated* if there exists $j' \in [m]$ such that

Algorithm 2: Solving (3) through binary search.

Initialization: $UB = \min_{x \in X} \max_{\xi \in \tilde{\Xi}} \xi^\top x$, $LB = \min_{x \in X} \min_{\xi \in \tilde{\Xi}} \xi^\top x$;
repeat
  $\rho = \frac{LB+UB}{2}$;
  Compute *cov*;
  Remove dominated solutions;
  Remove dominated scenarios;
  Compute $\omega := \{\min(16) : (12) - (14), (15), \alpha \geq 0\}$;
  if $\omega = 0$ **then** $LB \leftarrow \rho$;
  else $UB \leftarrow \rho$;
until $\frac{UB-LB}{UB} \leq \epsilon$;
Return: ρ

$cov_{sj} \geq cov_{sj'}$ for each $s \in [r]$. Removing dominated elements can significantly reduce the number of variables and constraints of the above MILP.

An optimal solution to problem (3) is recovered by solving $\{\min(16) : (12) - (14), (15), \alpha \geq 0\}$ after updating *cov* with the value of ρ returned by Algorithm 2. The solution vector z has K positive components each corresponding to a feasible solution $x_s \in X$.

2.4. Putting things together

The overall algorithm is comprised of three steps. First, a heuristic solution is computed using for instance, the heuristic presented in Appendix A or the one presented in Section 2.5. The heuristic returns an upper bound UB on the optimal solution of problem (2), as well as an initial feasible solution. The upper bound is used for generating only a subset of the set X, as explained in the following proposition.

PROPOSITION 4. *Assume that* $|X| > 1$, *let* $x' \in X$ *be such that* $\min_{\xi \in \Xi} \xi^\top x' \geq UB$, *and define* $v' := \min_{\mathbf{x} \in (X \setminus \{x'\})^K} \max_{\xi \in \Xi} \min_{k \in [K]} \xi^\top x^k$. *It holds that* $v' = v$.

Proof. We first remark that $v' \geq v$ since removing solutions from X creates a restriction of problem (2). Let $\bar{\mathbf{x}} = (\bar{x}^1, \ldots, \bar{x}^K)$ be an optimal solution to (2), with the corresponding worst-case scenario $\bar{\xi}$, and suppose that $v' > v$. Then, x' must be one of the K solutions

used in $\bar{\mathbf{x}}$, since otherwise we would have $v' = v$. Suppose, without loss of generality, that $\bar{x}^1 = x'$. We define $\hat{\mathbf{x}} = (\bar{x}^2, \bar{x}^2, \bar{x}^3, \ldots, \bar{x}^K)$. If $1 \notin \arg\min_{k\in[K]} \bar{\xi}^\top \bar{x}^k$, then $\min_{k\in[K]} \bar{\xi}^\top \hat{x}^k = \min_{k\in[K]} \bar{\xi}^\top \bar{x}^k$, proving $v' = v$. Otherwise, $1 \in \arg\min_{k\in[K]} \bar{\xi}^\top \bar{x}^k$ so that $v \geq UB$, a contradiction. $\square$

Following the above proposition, we define the set of non-dominated solutions as

$$X(UB) = \{x \in X : \exists \xi \in \Xi \text{ s.t. } \xi^\top x \leq UB\}.$$

The second step of our algorithm computes $X(UB)$. This can be done using a constraint-programming algorithm or a branch-and-bound algorithm, see Chassein et al. (2019). When the problem benefits from a particular structure, it can further be exploited to enumerate solutions quickly. For instance, the non-dominated solutions of the problems presented in our numerical experiments can be efficiently computed using dynamic programming algorithms. Each of the above algorithms (branch-and-bound, constraint programming, dynamic programming) relies on the computation of partial solutions, which can be defined as

$$X^{part}(UB) = \{I \subseteq [n] : \exists \xi \in \Xi, x \in X \text{ s.t. } \xi^\top x \leq UB, x_i = 1 \, \forall i \in I\}. \tag{17}$$

The condition $\xi^\top x \leq UB$ presented in (17) involves a complete solution $x \in X$ that is typically unknown when partial solution I is constructed. It is therefore necessary to replace $\xi^\top x$ by a possibly smaller expression depending only on I in order to eliminate non-relevant partial solutions as early as possible. With this in mind, we define $LB(I)$ as a lower bound on the cost required to complement I to a feasible solution in X, formally,

$$LB(I) \leq \min_{\substack{\xi\in\Xi, x\in X \\ x_i = 1 \, \forall i \in I}} \sum_{i\in[n]\setminus I} \xi_i x_i.$$

We remark that the right-hand-side in the above expression is a bilinear mixed-integer optimization problem whose exact solution can be very difficult. However, we do not need to solve this problem exactly but instead only to ensure that $LB(I)$ is not greater than its optimal value. In general, the computation of $LB(I)$ depends on both the structure of Ξ and X, and is therefore problem dependent. In Section 3, we provide examples of how this bound can be calculated in practice. Once $LB(I)$ is calculated, the condition $\xi^\top x \leq UB$ is then replaced by the weaker condition $\sum_{i\in I} \xi_i + LB(I) \leq UB$, where $LB(I)$ can be computed off-line in a pre-processing step. In our numerical experiments, we leverage these observations in order to reduce the number of solutions enumerated and alleviate the numerical burden of the enumeration step of our algorithm.

After generating $X(UB)$ in this manner, we finally call Algorithm 1 to solve problem (2).

2.5. A heuristic variant

The exactness of Algorithm 1 is crucially dependent on the enumeration of all *relevant* solutions, that is the set of solutions $X(UB)$ needs to contain *all* feasible solutions whose worst-case value is less than or equal to the value of the best solution known, UB. For combinatorial optimization problems that are loosely constrained, the cardinality of the set $X(UB)$ may therefore grow relatively fast, and the enumeration of this set may take a considerable amount of computational time. In this case, it is desirable to devise an algorithm that can provide good feasible solutions to problem (2) without enumerating the entire set $X(UB)$. Such an algorithm may be used on its own or in combination with our exact algorithm presented in Algorithm 1 so as to reduce the cardinality of the set $X(UB)$ (note that the lower the UB the smaller the cardinality of $X(UB)$). We next describe how our algorithm can be altered in order to work with a partially enumerated set of feasible solutions.

The main difference between this heuristic version and Algorithm 1 is the way lower bounds and candidate solutions are obtained. While the exact algorithm enumerates set X, here we assume that only a subset of solutions $\tilde{X}$ with $|\tilde{X}| < |X|$ is available. We therefore solve,

$$v(\tilde{X}, \tilde{\Xi}) := \min_{\mathbf{x} \in \tilde{X}^K} \max_{\xi \in \tilde{\Xi}} \min_{k \in [K]} \ \xi^\top x^k. \tag{18}$$

As a result of the solution of (18) with a partially enumerated set of solutions and scenarios, one obtains a solution $\bar{\mathbf{x}}$ as in Algorithm 1. This solution is sent to the separation problem to obtain a new scenario, ξ^*, as well as a potentially better upper bound and a new incumbent solution. The set $\tilde{X}$ is then enriched with the solution,

$$\tilde{x} \in \arg\min_{x \in X} {\xi^*}^\top x, \tag{19}$$

as well as its neighbors $N(\tilde{x})$, which is defined in a problem specific manner.

The heuristic algorithm just described is formally presented in Algorithm 3. We remark that problem (18) is not a relaxation of problem (2) as it restricts the set X. The value $v(\tilde{X}, \tilde{\Xi})$ is therefore not a true lower bound, and is merely used to establish a stopping criteria for the algorithm.

Solving (19) can be time-consuming for difficult combinatorial optimization problems. As the overall heuristic algorithm does not need an exact solution to (19), it can be

Algorithm 3: Heuristic scenario generation.

Initialization: $UB = \min_{x \in X} \max_{\xi \in \Xi} \xi^\top x$, $LB = \min_{x \in X} \min_{\xi \in \Xi} \xi^\top x$, $\tilde{\Xi} = \{\xi'\}$, $\bar{\mathbf{x}}' = null$;
repeat
  Compute $v(\tilde{X}, \tilde{\Xi})$ and $\bar{\mathbf{x}}$ by solving (18);
  $LB \leftarrow v(\tilde{X}, \tilde{\Xi})$;
  Compute $v(\bar{\mathbf{x}})$ by solving (4);
  if $v(\bar{\mathbf{x}}) < UB$ **then**
    $UB \leftarrow v(\bar{\mathbf{x}})$;
    $\bar{\mathbf{x}}' \leftarrow \bar{\mathbf{x}}$;
  end
  if $\frac{UB-LB}{LB} > \epsilon$ **then**
    $\tilde{\Xi} \leftarrow \tilde{\Xi} \cup \{\xi^*\}$;
    scenario_added = ***true***;
    Compute $\tilde{x}$ by solving (19), possibly heuristically;
    $\tilde{X} \leftarrow \tilde{x} \cup N(\tilde{x})$;
  end
  else *scenario_added* = ***false***;
until *scenario_added* = ***false***;
Return: $\bar{\mathbf{x}}'$

convenient to obtain instead a good feasible solution typically obtained through ad-hoc heuristic algorithms. In the numerical results presented in Section 3, we simply use a relax-and-fix heuristic that first solves the linear relaxation of problem (19). The algorithm then proceeds to fix the variables that take an integer value in the relaxation solution to their current values and solve the restricted problem (19) to return an integer feasible solution. In order to improve the quality of solutions returned by this relax-and-fix heuristic we impose that the relaxation solution should improve the best known integer solution for scenario ξ^* that is already in the set $\tilde{X}$. To this end, we impose the constraint ${\xi^*}^\top x \leq \min_{x \in \tilde{X}} {\xi^*}^\top x$ in solving the linear relaxation of (19).

3. Computational experiments

In this section, we present numerical results, using our scenario generation algorithm to solve various robust adaptive combinatorial optimization problems, namely, the shortest path problem and the knapsack problem with conflicts. We compare the numerical performance of our algorithm (denoted SG) to the existing methods from the K-adaptability literature, more specifically, the direct solution of the monolithic formulation of Hanasusanto et al. (2015) by an optimization solver (denoted M), and the branch-and-bound algorithm of Subramanyam et al. (2019) (denoted BB).

For each algorithm tested the time limit is set to two hours (7200 seconds), for our scenario generation algorithm the optimality tolerance ϵ is set to 0.005. All mixed integer linear programming and linear programming models are solved using IBM ILOG Cplex 12.8 solver with a single thread. The scenario generation algorithm and the monolithic formulation of Hanasusanto et al. (2015) are implemented with Julia 1.2.0 Bezanson et al. (2017) using the mathematical programming package JuMP Dunning et al. (2017). Their source code is available here. The branch-and-bound algorithm of Subramanyam et al. (2019) is adapted to each application using the authors' implementation available here.

All our experiments are conducted using a 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 2.5 GHz machine with 128Go RAM running Linux OS. The resources of this machine are strictly partitioned using Slurm Workload Manager[1] to run several tests in parallel. The resources available for each run (algorithm-instance) are set to two threads and a 20 Go RAM limit. This virtually creates 12 independent machines, each running one single instance at a time.

3.1. Shortest path problem

Setting The first problem on which we assess our algorithm is the adaptive shortest path problem, which has been previously studied in Hanasusanto et al. (2015) and Subramanyam et al. (2019). In this problem, K paths are prepared without exact knowledge of the arc costs, where the shortest among them is to be implemented once the actual arc costs are revealed. We provide numerical results on instances generated randomly according to the procedure proposed by Hanasusanto et al. (2015) and available online. As such, we consider a network (V, A) where the cost of each arc $(i, j) \in A$ is characterized as $\bar{c}_{ij} + \xi_{ij}\hat{c}_{ij}$,

[1] `https://slurm.schedmd.com/` (accessed June 2019)

with $\bar{c}_{ij}$ the nominal cost, and $\hat{c}_{ij}$ the maximal deviation. The uncertain parameter ξ can take any value within the budgeted uncertainty set $\Xi^{\Gamma} = \{\xi \in \{0,1\}^{|\mathcal{A}|} | \sum_{(i,j)\in\mathcal{A}} \xi_{ij} \leq \Gamma\}$. The feasibility set X is characterized by the classical flow constraints, $X = \{x \in \{0,1\}^{|A|} \mid \sum_{(i,j)\in\delta^+(i)} x_{ij} - \sum_{(j,i)\in\delta^-(i)} x_{ij} = b_i, \forall i \in V\}$, where $b_s = -1$, $b_t = 1$, and $b_i = 0$ for $V \setminus \{s,t\}$, and sets $\delta^+(i)$ and $\delta^-(i)$ represent the forward and backward stars of node $i \in V$, respectively. We use the heuristic solution of the bilinear formulation presented in Appendix A as an initial step of our algorithm. Previously published results Chassein et al. (2019) as well as preliminary experiments prove this heuristic to provide very good quality solutions in a short amount of computational time. Further, to calculate $LB(I)$ where I contains the indices of a path from s to u, we consider the shortest path from u to t using costs $\bar{c}$. We remark that the shortest path from each vertex $i \in V$ to t can be calculated in polynomial time in a preprocessing step. Notice that the results from Chassein et al. (2019) are not included in the comparison as they do not scale up with K. Specifically, the latter algorithm solves all instances within the time limit for $K \in \{2,3\}$ while not being able to solve any instance for $K \in \{4,5,6\}$.

Results In Figure 5, we present the percentage of instances solved to optimality by each solution method over hundred instances. We consider three different instance sizes $|V| \in \{20,25,30\}$, two different uncertainty budgets $\Gamma \in \{3,6\}$, and five different values of $K \in \{2,3,4,5,6\}$. From left to right, the first two graphics illustrate the evolution of the percentage of instances solved to optimality as K increases, presented as an average over the instance sizes, $|V| \in \{20,25,30\}$. The last two graphics illustrate the evolution of the same indicator as $|V|$ increases, presented as an average over the number of policies, $K \in \{2,3,4,5,6\}$. Figure 3 illustrates that most of the solution time of SG is spent within Algorithm 1, the heuristic algorithm and the computation of $X(UB)$ being very fast in all instance categories.

As it is clear from these graphics, the proposed scenario generation algorithm compares favorably to the existing methods in the literature, solving on average %85 of all instances for $\Gamma = 3$, and %50 of all instances for $\Gamma = 6$. Except for the case $K = 2$ and $\Gamma = 6$, the scenario generation method always has the most number of instances solved to optimality. It additionally scales reasonably well with increasing values of K, the number of policies, compared to the other two methods. It is also worth noting that among the three methods,

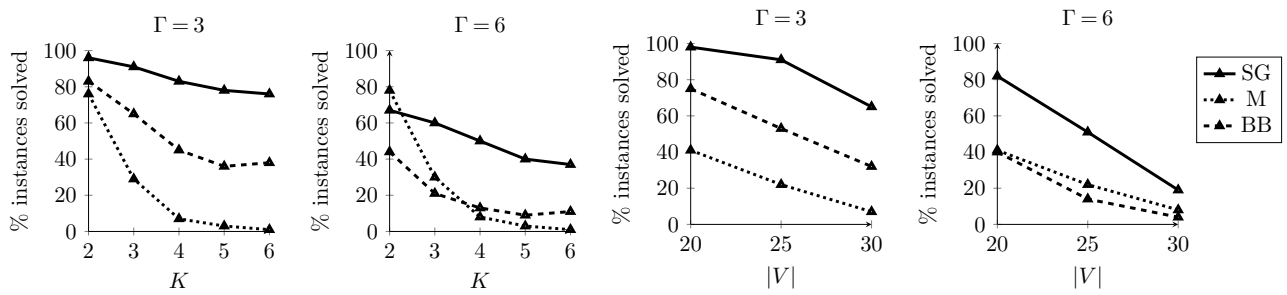

Figure 2 **Percentage of instances solved for the adaptive shortest path problem.**

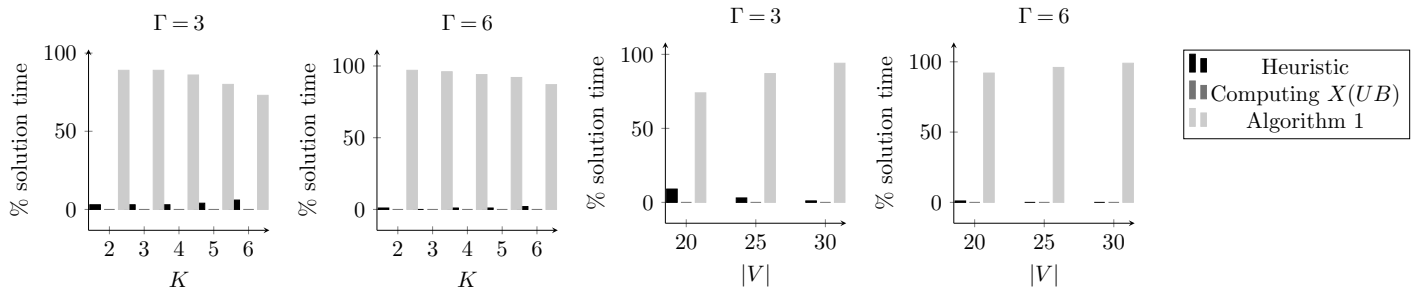

Figure 3 **Geometric averages of the time spent in each of the three steps of the scenario generation algorithm, considering only the instances solved to optimality.**

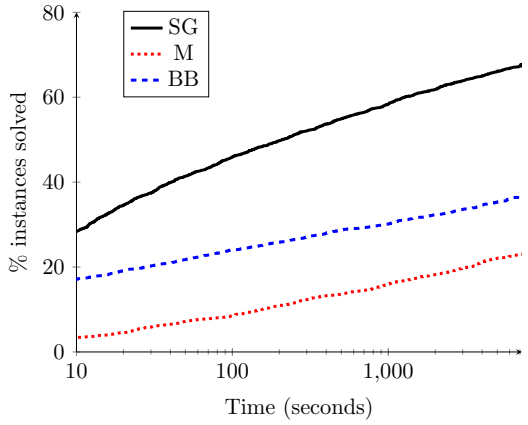

Figure 4 **Performance profile comparing the three methods for the adaptive shortest path problem.**

the monolithic approach suffers most severely from increasing values of K, as the number of binary variables resulting from the linearization of bilinear terms increases.

In Figure 4, we additionally present a performance profile that shows the percentage of instances solved to optimality by each method through time. This performance profile was obtained by aggregating all 3000 instances (100 instances for each combination of $|V|$, Γ, and K). Accordingly, the scenario generation method solved %67 of all instances

whereas BB solved %36 and M solved %23. Further at 10 seconds, M solves %3.4 of all instances whereas BB solves %17.1. To solve as many instances, SG takes 0.56 and 3.97 seconds, respectively. According to this comparison the scenario generation method is 17 times faster than M and 2.5 times faster than BB. A similar comparison performed at 100, 1000 and 7200 seconds reveals SG to be 48, 274, and 1222 times faster than M and 15, 82 and 290 times faster than BB, respectively.

3.2. Knapsack problem with conflicts

Setting We next study an adaptive knapsack problem with conflicts that is inspired by the capital budgeting problem and its variants studied in the $K-$adaptability literature Hanasusanto et al. (2015),Subramanyam et al. (2019). Here, the objective is to prepare K different combinations of items that respect the capacity and conflict constraints without exact knowledge of their profits. The most profitable among these K combinations is to be implemented upon actual realization of the uncertain profits.

Let $N = \{1, \ldots, n\}$ be the set of items considered. In this version of the problem, the feasibility set X can be characterized in different ways, which affects the efficiency of the resulting algorithms. Let w_i be the weight of an item, B be the knapsack budget and $C \subseteq N \times N$ be the set of conflicts. Herein we consider either the conflict formulation defined by the constraints $X^{\text{conf}} = \{x \in \{0,1\}^{|N|} \mid \sum_{i \in N} w_i x_i \leq B, x_i + x_j \leq 1 \quad \forall (i,j) \in C\}$, or the aggregated formulation defined by the constraints $X^{\text{agg}} = \{x \in \{0,1\}^{|N|} \mid \sum_{i \in N} w_i x_i \leq B, |E_i| x_i + \sum_{j \in E_i} x_j \leq |E_i| \quad \forall i \in N\}$ where $E_i = \{j \in N \mid (i,j) \in C \text{ or } (j,i) \in C\}$ is the set of items that are in conflict with i. The latter was proposed by Hifi and Michrafy (2007) and was used as the basis of heuristic algorithms for solving the knapsack with conflicts problem in Hifi (2014). The uncertain profits are characterized as $(1 + \sum_{j \in M} \frac{\Phi_{ij} \xi_j}{2}) \bar{p}_i$, where $\bar{p}_i$ is the nominal profit of item $i \in N$, $|M|$ is the number of uncertain factors, and $\Phi \in \mathbb{R}^{|N| \times |M|}$ is the factor loading matrix. The i-th row of Φ, denoted Φ_i, is characterized by the set $\{\Phi_i \in [-1,1]^{|M|} \mid \sum_{j \in M} |\Phi_{ij}| = 1\}$, whereas the uncertainty set Ξ is characterized as $[-1,1]^{|M|}$. As a result, the realized profit of each object $i \in N$ remains within the interval $[\bar{p}_i - \frac{\bar{p}_i}{2}, \bar{p}_i + \frac{\bar{p}_i}{2}]$.

The instances are generated randomly closely following the procedure proposed by Hanasusanto et al. (2015) and are available online. Specifically, the parameter $|M|$ is set to 4 and we generate 10 instances for each instance size $|N| \in \{100, 150, 200, 250, 300, 350\}$: w_i

is uniformly generated in $[0, 100]$, $B = \sum_{i \in N} w_i/2$, $\bar{p}_i = c_i/5$, and Φ_i is uniformly generated in $[-1, 1]^4$. For each instance, we additionally introduce randomly generated conflicts between items. To do so, we use conflict graphs parametrized by a given density inspired by the instances used in Sadykov and Vanderbeck (2013) in the context of the bin packing problem with conflicts. In this context, the density of a graph $G = (N, E)$ is defined as the ratio between $|E|$ and the cardinality of the edge set of the complete graph having the same number of vertices. In our numerical experiments, we consider conflict graphs with density 0.5.

Regarding $LB(I)$, a strong lower bound of the bilinear mixed-integer problem that can be computed efficiently is not immediately available, because the set $[-1, 1]^{|M|}$ does not have a clear dominated scenario, unlike the aforementioned budgeted uncertainty set. For this reason, we only perform the optimality check when the full solution x is known. In this case, $LB(I) = 0$ and $x \in X^{part}(UB)$ only if $\min_{\xi \in \Xi} \xi^\top x \leq UB$.

Results for the heuristic algorithms We first present an evaluation of different heuristic approches that can be used as the first step of our algorithm. Our preliminary results showed that the heuristic resolution of the bilinear formulation presented in Appendix A was too slow for this problem, taking nearly two hours of computational time for instances with 100 items only. This behavior is due to the optimization over variables $(\alpha, \mathbf{x})$, which takes too much time as N grows. We therefore focus on the two other approaches: the heuristic version of our algorithm presented in Section 2.5, and the heuristic version of the branch-and-bound algorithm of Subramanyam et al. (2019) where $k-$adaptability problems are solved in sequence for $k = 1, \ldots, K$, and the optimal solution of the $(k-1)-$adaptable problem is fixed in the $k-$adaptable problem.

For this comparison, we consider the scenario generation algorithm SG and the branch-and-bound algorithm BB, with two formulations X^{agg} and X^{conf}. For each of the four methods, we compare the relative distance of the upper bound provided by the algorithm to the true optimal value of each instance obtained through our exact algorithm SG. In Figure 5, we represent a scatter plot with this relative gap along with the solution time of each algorithm for each value of $K \in \{2, 3, 4, 5, 6\}$ and $|N| = 300$. The detailed results for instances of size $|N| \in \{100, 150, 200, 250, 300\}$ are given in Tables 1 and 2 in Appendix B. Comparing the results of the two formulations for BB displayed on Figure 5, we realize that while the solution times may differ based on the formulation, the solution obtained

does not change. Similar results were obtained for smaller instances and for this reason, the details of hBB are only provided with X^{conf} in Appendix B. This behavior is due to the fact that hBB solves an MILP to exact optimality at every iteration, so in the absence of multiple optima the same optimal solutions are returned irrespective of the formulations used. On the other hand, the heuristic variant of the scenario generation algorithm solves (19) heuristically within the algorithm and therefore the solution quality depends on the linear relaxation of the respective formulation.

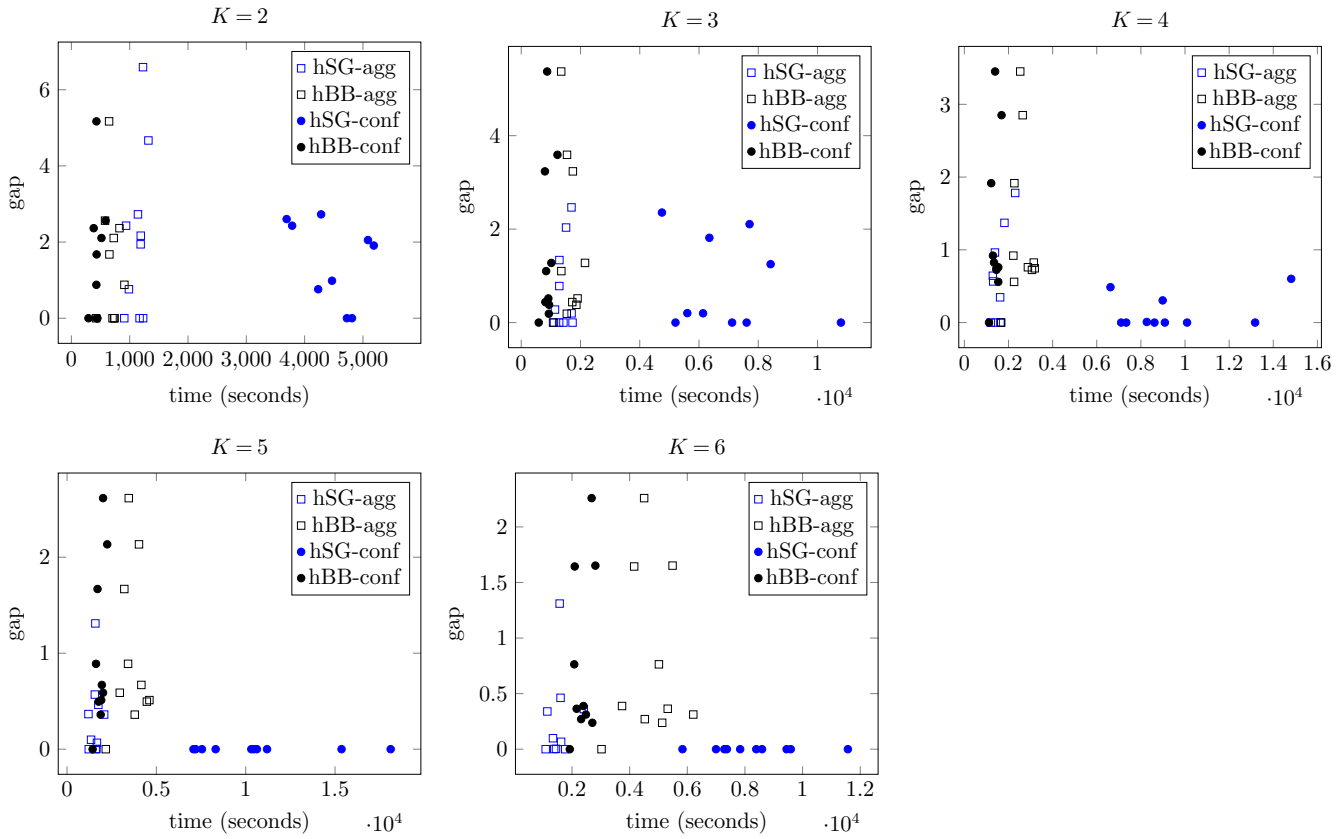

Figure 5 **Solution times and ratio to the best solution for the instances with 300 items.**

The results presented in Figure 5 lead to the conclusion that the heuristic scenario generation algorithm with the formulation X^{conf}, hSG-agg, leads to better quality solutions, especially as K increases. However, this algorithm is costly in terms of computation time which prohibits its utilisation for larger instances. Further, among the remaining three methods, hBB-conf and hSG-agg, are compatible in terms of the solution time required, while the solution time for hBB-agg increases significantly with K. Therefore, hBB-conf

and hSG-agg, emerge as computationally viable heuristic methods as N and K increase. Among these two methods hSG-agg provides consistently good quality solutions.

Results for the exact algorithms We test all three methods for the two formulations X^{conf} and X^{agg} on the aforementioned instances. In Figure 6, we present the number of instances solved to optimality by each solution method over ten instances. The graphic on the left illustrates the evolution of the number of instances solved to optimality as K increases, presented as an average over the instance sizes, $|N| \in \{100, 150, 200, 250, 300, 350\}$. The graphic on the right illustrates the evolution of the same indicator as $|N|$ increases, presented as an average over the number of policies, $K \in \{2, 3, 4, 5, 6\}$.

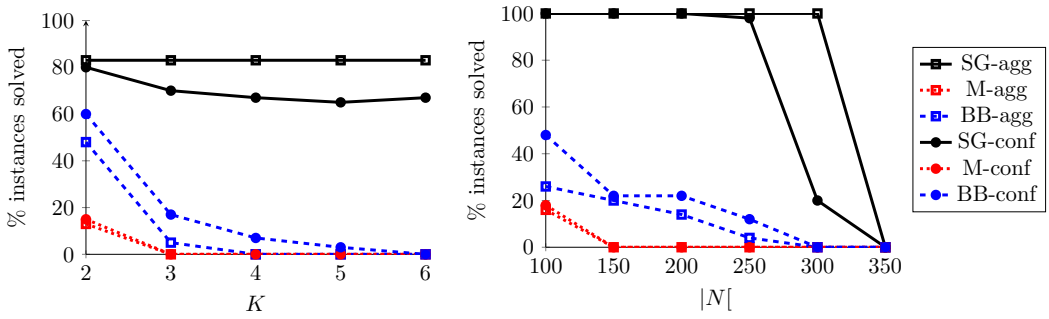

Figure 6 **Instances solved for the adaptive knapsack problem.**

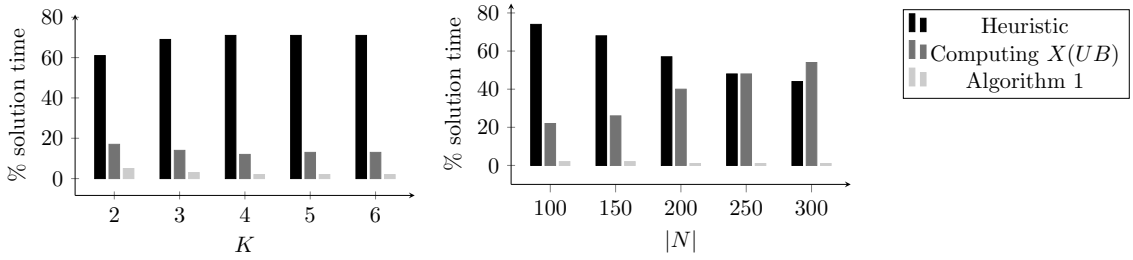

Figure 7 **Geometric averages of the time spent in each of the three steps of the scenario generation algorithm based on the aggregated formulation, considering only the instances solved to optimality.**

As illustrated by these results, the comparative efficiency of the three algorithms is not impacted by the underlying formulations. The monolithic approach M was able to solve only a small percentage of the smallest instances with 100 items, regardless of the underlying formulation. The branch-and-bound algorithm BB was more effective and more

sensitive to the underlying formulation than M. Its implementation with X^{conf} solved %60 of all instances with $K = 2$, while it failed to scale with increasing values of K. When comparing formulations, BB was able to solve more instances in each category coupled with the conflict formulation X^{conf} than the aggregated formulation X^{agg}. The scenario generation algorithm SG coupled with the aggregated formulation X^{agg} was able to solve all instances up to $|N| = 300$. SG performed slightly worse with X^{conf}, solving to optimality all instances up to $|N| = 250$. For SG, the inability to solve the largest instances with X^{conf} is due to a combination of the heuristic and enumerative steps, which no longer succeed in enumerating $X(UB)$ within the time limit as illustrated in Figure 7. A closer inspection of the heuristic algorithm reveals that the solution of (19) takes longer using this formulation. For BB, the formulation X^{conf} provides a better performance as its relaxation is stronger than X^{agg}. These results highlight the particularities of branch-and-bound and scenario generation algorithms. The scenario generation algorithm is more sensitive to the time spent in the heuristic step and the quality of the upper bound obtained as this affects directly the number of solutions that should be enumerated. On the other hand, the branch-and-bound algorithm is more sensitive to the mathematical formulation, the strength of its linear relaxation and the required solution time being important factors.

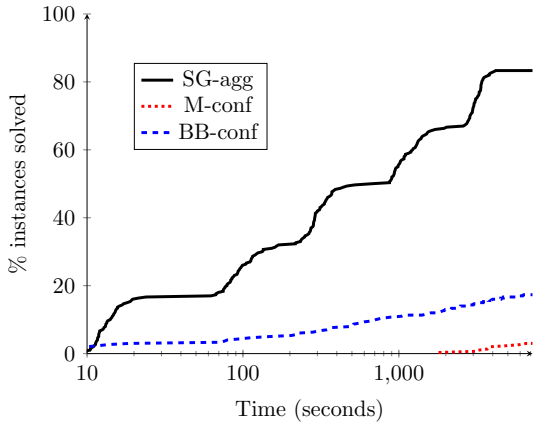

Figure 8 **Performance profile comparing the three methods for the adaptive knapsack problem with conflicts.**

In Figure 8, we additionally present a performance profile that shows the percentage of instances solved to optimality by each method through time. This performance profile was obtained by aggregating all 300 instances (10 instances for each combination of N and K) for the best formulation for each algorithm, that is, SG-agg, M-conf, and BB-conf.

Accordingly, the scenario generation method solved %83.3 of all instances whereas BB solved %17.3 and M solved %3. Further at 100 seconds, M solves %0 of all instances whereas BB solves %4.3. To solve as many instances as BB, SG takes 11.67 seconds. According to this comparison the scenario generation method is 8.5 times faster than the branch-and-bound algorithm of Subramanyam et al. (2019). A similar comparison performed at 1000 and 7200 seconds reveals SG 69 and 107 times faster than BB. At 7200 seconds, the scenario generation algorithm is 637 times faster than the monolithic approach M.

4. Concluding remarks

In this paper, we present a new exact algorithm for solving the min-max-min robust combinatorial optimization problem. The algorithm combines the enumeration scheme presented in Chassein et al. (2019) with a scenario-based relaxation, reformulated as a p-center problem and solved through a binary-search algorithm. While the original work was limited to $K \in \{2, 3\}$ and budget uncertainty, the algorithm proposed herein scales reasonably well with K and is able to solve more instances to optimality than the state-of-the-art algorithms from the $K-$adaptability literature Hanasusanto et al. (2015), Subramanyam et al. (2019), in particular for large values of K. The main advantage of our approach is that it is little impacted by the complexity of solving the underlying deterministic problem. This is illustrated for the knapsack problem with conflicts, for which our algorithm can solve all instances with up to 300 items while Hanasusanto et al. (2015) and Subramanyam et al. (2019) can only solve a small part of the instances with up to 100 and 250 items, respectively. We additionally propose a heuristic variant of our algorithm based on a partial enumeration of good feasible solutions that can be used when enumerating all good feasible solutions is computationally prohibitive and it is hard to optimize over the set X. One the one hand, when optimizing over X is easy, as for the shortest path problem, our new heuristic cannot compete with the one from Chassein et al. (2019). On the other hand, the new heuristic behaves well for harder problems and even compares favorably to the heuristic variant of the branch-and-bound algorithm from Subramanyam et al. (2019) on the knapsack problem with conflicts for the larger values of K (the heuristic from Chassein et al. (2019) being able to cope only with the smallest instances for that problem).

References

Agra A, Santos MC, Nace D, Poss M (2016) A dynamic programming approach for a class of robust optimization problems. *SIAM Journal on Optimization* 26(3):1799–1823.

Aissi H, Bazgan C, Vanderpooten D (2009) Min-max and min-max regret versions of combinatorial optimization problems: A survey. *Eur. J. Oper. Res.* 197(2):427–438.

Arslan A, Detienne B (2020) Decomposition-based approaches for a class of two-stage robust binary optimization problems Available at https://hal.inria.fr/hal-02190059/.

Ayoub J, Poss M (2016) Decomposition for adjustable robust linear optimization subject to uncertainty polytope. *Comput. Manag. Science* 13(2):219–239.

Ben-Tal A, El Ghaoui L, Nemirovski A (2009) *Robust optimization*, volume 28 (Princeton University Press).

Bertsimas D, Brown DB, Caramanis C (2011) Theory and applications of robust optimization. *SIAM review* 53(3):464–501.

Bertsimas D, Dunning I (2016) Multistage robust mixed-integer optimization with adaptive partitions. *Oper. Res.* 64(4):980–998.

Bertsimas D, Dunning I, Lubin M (2016) Reformulation versus cutting-planes for robust optimization. *Comput. Manag. Science* 13(2):195–217.

Bertsimas D, Georghiou A (2018) Binary decision rules for multistage adaptive mixed-integer optimization. *Math. Program.* 167(2):395–433.

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

Bezanson J, Edelman A, Karpinski S, Shah VB (2017) Julia: A fresh approach to numerical computing. *SIAM review* 59(1):65–98, URL `https://doi.org/10.1137/141000671`.

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

Buchheim C, Kurtz J (2018a) Complexity of min–max–min robustness for combinatorial optimization under discrete uncertainty. *Discrete Optimization* 28:1–15.

Buchheim C, Kurtz J (2018b) Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO J. Comput. Optim.* 6(3):211–238, URL `http://dx.doi.org/10.1007/s13675-018-0103-0`.

Chalermsook P, Cygan M, Kortsarz G, Laekhanukit B, Manurangsi P, Nanongkai D, Trevisan L (2017) From gap-eth to fpt-inapproximability: Clique, dominating set, and more. *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 743–754 (IEEE).

Chassein A, Goerigk M (2020) On the complexity of min–max–min robustness with two alternatives and budgeted uncertainty. *Discrete Applied Mathematics* .

Chassein AB, Goerigk M, Kurtz J, Poss M (2019) Faster algorithms for min-max-min robustness for combinatorial problems with budgeted uncertainty. *European Journal of Operational Research* 279(2):308–319.

Chen D, Chen R (2009) New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems. *Computers & Operations Research* 36(5):1646–1655.

Contardo C, Iori M, Kramer R (2019) A scalable exact algorithm for the vertex p-center problem. *Computers & OR* 103:211–220.

Daskin MS (2011) *Network and discrete location: models, algorithms, and applications* (John Wiley & Sons).

Dunning I, Huchette J, Lubin M (2017) Jump: A modeling language for mathematical optimization. *SIAM Review* 59(2):295–320, URL `http://dx.doi.org/10.1137/15M1020575`.

Eufinger L, Kurtz J, Buchheim C, Clausen U (2019) A robust approach to the capacitated vehicle routing problem with uncertain costs. *INFORMS Journal on Optimization* To appear.

Fischetti M, Monaci M (2012) Cutting plane versus compact formulations for uncertain (integer) linear programs. *Math. Program. Comput.* 4(3):239–273.

Gabrel V, Murat C, Thiele A (2014) Recent advances in robust optimization: An overview. *European Journal of Operational Research* 235(3):471–483.

Goerigk M, Kurtz J, Poss M (2020) Min-max-min robustness for combinatorial problems with discrete budgeted uncertainty. *Discrete Applied Mathematics* In press.

Hanasusanto GA, Kuhn D, Wiesemann W (2015) K-adaptability in two-stage robust binary programming. *Operations Research* 63(4):877–891.

Hifi M (2014) An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem. *Engineering Optimization* 46(8):1109–1122.

Hifi M, Michrafy M (2007) Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & operations research* 34(9):2657–2673.

Kouvelis P, Yu G (2013) *Robust discrete optimization and its applications*, volume 14 (Springer Science & Business Media).

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

Minieka E (1970) The m-center problem. *Siam Review* 12(1):138–139.

Pessoa AA, Poss M (2015) Robust network design with uncertain outsourcing cost. *INFORMS J. Comput.* 27(3):507–524.

Poss M (2018) Robust combinatorial optimization with knapsack uncertainty. *Discrete Optimization* 27:88–102.

Sadykov R, Vanderbeck F (2013) Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing* 25(2):244–255.

Subramanyam A, Gounaris CE, Wiesemann W (2019) K-adaptability in two-stage mixed-integer robust optimization. *Mathematical Programming Computation* 1–32.

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

Appendix A: Dualization

We present next the dualized reformulation proposed by Hanasusanto et al. (2015) as well as its heuristic variant first suggested by Chassein et al. (2019). Let us define Ξ as the polytope $\Xi := \{A\xi \leq b, \xi \geq 0\}$. Then, applying an epigraph reformulation to problem (2), we have

$$\min_{\mathbf{x}\in X^K} \left\{ \max \eta : \xi \in \Xi,\, \eta \leq \xi^\top x^k,\, k \in [K] \right\}. \tag{20}$$

Introducing the vectors of dual variables α and β for the constraints $A\xi \leq b$ and $\eta \leq \xi^\top x^k$, $k \in [K]$, we can dualize the inner linear program of (20) to obtain

$$\begin{aligned} \min \quad & b^\top \alpha \\ \text{s.t.} \quad & \alpha^\top A - \sum_{k\in[K]} \beta_k x^k \geq 0 \\ & \sum_{k\in[K]} \beta_k \geq 1 \\ & x^1, \ldots, x^K \in X \\ & \alpha, \beta \geq 0. \end{aligned}$$

The above non-linear mixed-integer linear program is addressed heuristically in Chassein et al. (2019), by iteratively optimizing over $(\alpha, \mathbf{x})$ and (α, β) until no further improvement is obtained. Alternatively, Hanasusanto et al. (2015) apply the McCormick linearization to each product $\beta_k x^k$ to obtain an exact MILP reformulation.

Appendix B: Heuristic results

	100			150			200			250			300		
K	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf
2	9	20	2	52	94	8	149	311	32	474	1649	120	1130	4574	439
3	10	24	3	73	132	17	182	402	66	539	2450	258	1416	6968	894
4	10	26	5	96	181	25	207	505	101	685	3236	399	1564	9410	1403
5	10	25	7	89	170	35	206	565	139	638	3687	577	1565	10621	1855
6	10	25	8	78	172	44	198	529	187	625	3196	730	1537	8292	2365

Table 1 Average solution time for each heuristic algorithm in seconds.

	100			150			200			250			300		
K	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf	hSG-agg	hSG-conf	hBB-conf
2	1.0	0.7	1.8	2.5	2.2	1.9	1.2	1.2	1.4	1.3	2.3	1.1	2.1	1.8	1.5
3	0.5	0.3	1.3	1.4	1.0	1.9	0.2	0.4	1.0	1.0	1.0	1.3	0.7	0.8	1.6
4	0.5	0.0	0.5	0.4	0.3	1.4	0.3	0.2	0.5	0.6	0.3	0.8	0.6	0.1	1.3
5	0.5	0.0	0.1	0.2	0.0	0.7	0.2	0.0	0.3	0.3	0.0	0.4	0.3	0.0	1.0
6	0.5	0.0	0.0	0.4	0.0	0.2	0.2	0.0	0.1	0.3	0.0	0.2	0.3	0.0	0.8

Table 2 Average relative optimality gap obtained with each heuristic algorithm.