Column Generation in Column-and-Constraint Generation for Adjustable Robust Optimization with Interdiction-Type Linking Constraints

Henri Lefebvre, Martin Schmidt, Johannes Thürauf

Abstract. Adjustable robust optimization (ARO) is a powerful tool to model problems that have uncertain data and that feature a two-stage decision making process. Computationally, they are often addressed using the column-and-constraint generation (CCG) algorithm introduced by Zeng and Zhao (2013). While it was empirically shown that the algorithm scales well if all second-stage decisions are continuous, the presence of integer variables in the second stage rapidly leads to challenging large-scale mixed-integer problems within CCG. These problems can no longer be solved to global optimality within reasonable time limits in general. In this work, we explicitly focus on ARO problems with mixed-integer second-stage decisions and discuss the main difficulties of successfully applying CCG to this problem class. We then introduce, for a large set of problems with specific structural properties, a stronger formulation, which can be used in place of the master problem in the classic CCG algorithm. We show how this model can be effectively solved by column generation (CG). Additionally, we introduce a new CG-based heuristic that is able to generate new feasible points to speed up the overall method. We apply this nested scheme, combining CCG and CG, to three problems from logistics and scheduling. The numerical results show that the proposed method significantly outperforms the classic CCG.

1. Introduction

In the last decades, robust optimization (RO) has emerged as a powerful tool to tackle uncertain parameters in optimization problems. In contrast to probabilistic approaches, very limited knowledge about the underlying distribution of the uncertain data is required. Indeed, only a subset of the support of the density function is needed to define a so-called uncertainty set. This set is then viewed as gathering scenarios with equally unknown probability against which each decision is evaluated from a worst-case perspective. Successful applications of RO are plethora and can be found in transportation (Agra et al. 2013), finance (Xidonas et al. 2020), energy systems (Sun and Conejo 2021), and machine learning (Ben-Tal et al. 2011), among others.

While situations in which all decisions have to be taken "here and now", i.e., before the realization of the uncertainty, are well understood, see, e.g., Ben-Tal et al. (2009) and Gabrel et al. (2014), problems featuring a two-stage decision process are considerably harder to analyze both from a theoretical as well as from a practical point of view. In such problems, there is an additional decision-making phase after the unknown parameters have been revealed. This leads to so-called "second-stage decisions", which actually depend on the revealed uncertainty. Following the

Date: January 30, 2025.

2020 *Mathematics Subject Classification.* 90-XX, 90C11, 90C17, 90C57.

Key words and phrases. Adjustable robust optimization, Column generation, Exact methods, Optimization under uncertainty, Two-stage optimization.

standard terminology of ARO, we also refer to second-stage decisions as "wait-and-see decisions."

To the best of our knowledge, such ARO problems have been, for the first time, considered by Ben-Tal et al. (2004), who also showed NP-hardness for ARO problems, even in the case of continuous wait-and-see decisions. For that reason, a vast majority of works considered problems with continuous second-stage decisions only. Under this assumption, Ben-Tal et al. (2004) introduced the idea of affine decision rules (ADR), which consists in restricting the second-stage decisions to affinely depend on the uncertain parameters. Doing so, they obtained "good" tractable approximations for problems with fixed recourse, i.e., problems without matrix uncertainty. Later, the approach has been extended to problems without fixed recourse when the uncertainty set is an ellipsoid; see Ben-Tal et al. (2009). Unfortunately, such ADRs generally lead to feasible points, which may not be globally optimal for the fully adjustable problem; see, e.g., Bertsimas et al. (2010) or Bertsimas and Goyal (2011) for optimality conditions of ADRs.

Further, the ADR approach cannot be applied to problems with integer restrictions on the second-stage variables. To fill this gap, piecewise static decision rules, also known as K-adaptability, have been introduced by Bertsimas and Caramanis (2010). Further algorithms have been developed in Hanasusanto et al. (2015) and Subramanyam et al. (2019). However, these methods do not guarantee global optimality in general; see Kurtz (2023) for approximation guarantees.

Regarding exact approaches, Zhen et al. (2018) suggested to use Fourier–Motzkin elimination to address ARO problems with a small number of continuous second-stage decisions. For larger instances, the most prominent approach has been introduced by Zeng and Zhao (2013), and is referred to in the literature as the column-and-constraint generation (CCG) algorithm. In a nutshell, it iterates between the solution of a (relaxed) master problem, in which only a limited number of scenarios is considered, and a set-augmentation step, in which missing scenarios are identified and added to the master problem. Because second-stage decisions depend on the scenario in which they are taken, new variables and constraints must be added for each scenario, hence the name of the method. The algorithm was empirically shown to be "quite scalable" for a large variety of problems in which second-stage decisions are continuous; see Bertsimas and Shtern (2018).

While the extension of CCG to problems with mixed-integer second-stage decisions is possible, see Zhao and Zeng (2012), it becomes impractical quickly even for medium-sized instances. The reason for this is the following: At each iteration of the algorithm, a new set of integer variables is added to the master problem. This rapidly leads to a large-scale mixed-integer program (MIP) for which finding good lower and upper bounds gets harder and harder.

Under the assumption that only the objective function is uncertain and all here-and-now decisions are binary, Kämmerling and Kurtz (2020) introduced a branch-and-cut algorithm. Later, a similar approach based on CG was developed by Arslan and Detienne (2022) with some restrictive assumptions on the linking constraints between the here-and-now and the second-stage variables. In Detienne et al. (2024), the authors extend this approach to more general but convex linking constraints.

In this paper, we explicitly consider problems with mixed-integer second-stage decisions and adapt the CCG algorithm to alleviate the discussed burdens. Our main contributions are as follows:

(i) We shed light on the main weaknesses of CCG in the presence of mixed-integer second-stage decisions.

(ii) We introduce a new extended formulation for ARO problems with a specific structure, which can be used to derive a new CCG-type algorithm. In this new formulation, the wait-and-see decisions are no longer integer but continuous variables and do not require any branching.
(iii) We discuss how to solve the new extended formulation, which can be used as the master problem in CCG, by using CG techniques and branch-and-price (B&P).
(iv) We introduce a new upper-bounding technique, which can be used throughout the branch-and-price algorithm.
(v) We computationally assess the performance of the new CCG algorithm on three applications in logistics and in job scheduling.

The remainder of this paper is organized as follows. In Section 2, we formally state the class of problems considered in this paper and discuss the main assumptions. In Section 3, we recall the classic CCG scheme and analyze its main issues in presence of mixed-integer second-stage variables. Our main contribution, which is a stronger formulation of the CCG master problem, is presented in Section 4. Methods based on CG are also discussed in this section. In Section 5, we report on two computational experiments showing the superiority of our approach compared to the classic CCG algorithm before we finally conclude in Section 6.

Notation. For a convex set S, we denote its set of extreme points by $\operatorname{ext}(S)$. The set of extreme rays of S is denoted by $\operatorname{ray}(S)$. For $X := S \cap (\mathbb{R}^s \times \mathbb{Z}^t)$, we denote by $\operatorname{cont}(X) := S$ its continuous relaxation, i.e., the set obtained by dropping all integrality requirements. The smallest convex set, which contains X is called its convex hull and is denoted by $\operatorname{conv}(X)$, its closure is called $\operatorname{cl}(\operatorname{conv}(X))$. We let $\operatorname{cone}(X)$ denote the conic hull of X, i.e., the smallest convex cone containing X.

For a given set C, we let $M(C)$ be the set of all Lebesgue–Stieltjes measures defined on C with a suitable σ-algebra. For a function $f : \mathbb{R}^n \to \mathbb{R}$, its convex envelope is denoted by $\operatorname{vex}(f)$. For a set of indices $I \subseteq \{1, \dots, n\}$ and a vector $x \in \mathbb{R}^n$, we let x_I denote the vector consisting of the components of x with indices in I (in the same order).

2. Problem Statement

Let $X \subset \mathbb{R}^{n_x}$ be a given set of feasible decisions to be taken here and now and let $\Xi \subset \mathbb{R}^{n_\xi}$ be a given uncertainty set. For any here-and-now decision $x \in X$ and any scenario $\xi \in \Xi$, we denote by $Y(x, \xi) \subset \mathbb{R}^{n_y}$ the set of feasible second-stage decisions. Additionally, $c \in \mathbb{R}^{n_x}$ and $d(\xi) \in \mathbb{R}^{n_y}$, $\xi \in \Xi$, are cost vectors for the here-and-now and wait-and-see decisions.

We consider the class of problems given by

$$v^* := \min_{x \in X} \left\{ c^\top x + \sup_{\xi \in \Xi} \inf_{y \in Y(x,\xi)} d(\xi)^\top y \right\}. \tag{ARO}$$

For the ease of presentation, we assume that (ARO) always has a solution if the problem is feasible. Moreover, we suppose that the following assumptions are satisfied.

Assumption 1. *Problem (ARO) is bounded, i.e., $v^* > -\infty$.*

Assumption 2. *The uncertainty set Ξ is compact and its convex hull is a polyhedron with rational extreme points.*

Assumption 3. *For any $\hat{x} \in X$, the value function*

$$\xi \mapsto \mathcal{Q}(\hat{x}, \xi) := \inf \left\{ d(\xi)^\top y \colon y \in Y(\hat{x}, \xi) \right\}$$

is quasi-convex or the uncertainty set Ξ is discrete.

Assumption 4. *For any $\hat{x} \in X$, an oracle for solving the adversarial problem*

$$\max_{\xi \in \Xi} \inf_{y \in Y(\hat{x},\xi)} d(\xi)^\top y \tag{1}$$

is at hand.

Note that the latter assumptions allow for writing "$\max_\xi$" instead of "$\sup_\xi$", which we do in what follows.

These are classic assumptions for CCG algorithms to ensure finite termination; see, e.g., Zhao and Zeng (2012), Subramanyam (2022) and Lefebvre and Subramanyam (2024). Assumption 1 is a mild assumption for problems in which the objective function relates to costs or physical quantities. Note that we do not assume that the problem is feasible. Assumption 2 discards convex but nonlinearly described uncertainty sets from being used, yet allows for discrete, polyhedral, mixed-integer uncertainty sets as well as uncertainty sets defined as unions of polyhedral sets. Assumption 3 is a technical assumption, which is naturally satisfied by most of the practical applications in ARO; see Zhao and Zeng (2012). The latter is true because the value function of the second-stage problem typically is non-increasing in ξ. Assumption 4 states that an oracle is available for identifying the worst-case scenario in response to a given here-and-now decision. Note that the adversarial problem (1) is a bilevel problem for which, in general, any suitable bilevel solver can be used. This includes general mixed-integer bilevel solvers such as MibS (Tahernejad et al. 2020). Note, however, that these solvers require that all upper-level decisions, here ξ, that actually parameterize Y are bounded integers. Hence, while the theoretical developments of this paper apply to both the mixed-integer and purely continuous case for the uncertainty sets, no code is available today for the latter case.

We now state two additional and non-standard assumptions, which we need for our approach.

Assumption 5. *There exists an index set $I \subseteq \{1,\dots,n_x\}$ such that $x \in X$ implies $x_I \in \{0,1\}^{|I|}$ and that, for any $\hat{\xi} \in \Xi$, there exists a set $Z(\hat{\xi})$ such that*

$$Y(x,\hat{\xi}) = Z(\hat{\xi}) \cap \{y \in \mathbb{R}^{n_y} : 0 \le y_i \le u_i(1-x_i),\ i \in I\}, \tag{2}$$

for some vector u_I. For simplicity, we assume $|I| \le n_y$.

Assumption 6. *For any $\hat{x} \in X$ and any*

$$\xi^* \in \arg\max\{\inf\{d(\xi)^\top y : y \in Y(\hat{x},\xi)\} : \xi \in \Xi\}$$

returned by the oracle, the convex hull of $Z(\xi^*)$ is closed.

Assumption 5 states that the only linking constraints between the here-and-now and the wait-and-see decisions are interdiction-type constraints. These constraints naturally arise in a large variety of applications in which, here-and-now, strategic decisions have to be made to allow or interdict future decisions. This assumption is similar to the one made by Arslan and Detienne (2022) for cost-uncertain problems. Without loss of generality, we additionally assume that $0 \le y_I \le u_I$ is part of the definition of $Z(\xi)$. Assumption 6 is necessary for the correctness of our main reformulation and is very mild, as explained in the following remarks.

Remark 1. *Let $W : \mathbb{R}^{n_\xi} \to \mathbb{R}^{m_y \times n_y}$ and $h : \mathbb{R}^{n_\xi} \to \mathbb{R}^{m_y}$ be given $\mathbb{Q}$-linear mappings, i.e., there exists matrices $W^1,\dots,W^{n_\xi} \in \mathbb{Q}^{m_y \times n_y}$ and $h^1,\dots,h^{n_\xi} \in \mathbb{Q}^{m_y}$ such that*

$$W(\xi) := \sum_{k=1}^{n_\xi} \xi_k W^k \quad \text{and} \quad h(\xi) := \sum_{k=1}^{n_\xi} \xi_k h^k,$$

and assume that $Y(x,\xi)$ can be written as in (2) with

$$Z(\xi) = \left\{y \in \mathbb{Z}^{p_y} \times \mathbb{R}^{n_y - p_y} : W(\xi)y \ge h(\xi)\right\} \tag{3}$$

for any $\xi \in \Xi$. By Assumptions 1–3, it holds that, for any $\hat{x} \in X$, at least one solution to the adversarial problem is an extreme point of the convex hull of Ξ. Assumption 2 implies that these extreme points are rational. Thus, there always exists

$$\xi^* \in \mathbb{Q}^{n_\xi} \cap \arg\max\{\inf\{d(\xi)^\top y : y \in Y(\hat{x}, \xi)\} : \xi \in \Xi\}.$$

By further assuming that such a point is returned by the oracle in Assumption 4, we have that $Z(\xi)$ can be expressed as a MIP with rational entries and thus, its convex hull is closed; see Meyer (1974).

Remark 2. *Assume that $Z(\xi)$ is given as in (3) and that Ξ is a set of rational points, possibly described by a set of linear constraints together with integer requirements, then Assumption 6 is satisfied.*

3. Column-and-constraint Generation With Integer Decisions

The column-and-constraint generation algorithm has been introduced by Zeng and Zhao (2013) for robust problems in which second-stage decisions are continuous. Later, it has been extended to the mixed-integer case by Zhao and Zeng (2012) under Assumptions 1–3 by, notably, suggesting a way of fulfilling Assumption 4 in a generic way. For the sake of completeness, we briefly recall the formal statement of the algorithm as well as the key arguments for its finite termination.

3.1. **Extended Formulation.**

The starting point of the CCG algorithm is a so-called "extended formulation" of (ARO), which we state first. To this end, observe that by Assumption 2 and 3, (ARO) is equivalent to

$$\min_{x \in X} c^\top x + \max_{\xi \in \hat{\Xi}} \inf_{y \in Y(x,\xi)} d(\xi)^\top y, \tag{4}$$

with $\hat{\Xi} := \operatorname{vert}(\operatorname{conv}(\Xi))$. Assumption 2 implies $|\hat{\Xi}| < \infty$ and the finite maximum can be expressed by a finite set of constraints, leading to the model

$$\begin{aligned} \min_{x,\theta} \quad & c^\top x + \theta \\ \text{s.t.} \quad & x \in X, \\ & \theta \geq \inf_{y \in Y(x,\xi)} d(\xi)^\top y, \quad \xi \in \hat{\Xi}. \end{aligned}$$

The extended formulation of (ARO) is obtained by introducing new variables $y_\xi \in Y(x, \xi)$ for each $\xi \in \hat{\Xi}$. Thus, the extended model reads

$$\min_{x,\theta} \inf_{y_\xi} \quad c^\top x + \theta \tag{5a}$$
$$\text{s.t.} \quad x \in X, \tag{5b}$$
$$\theta \geq d(\xi)^\top y_\xi, \quad \xi \in \hat{\Xi}, \tag{5c}$$
$$y_\xi \in Y(x, \xi), \quad \xi \in \hat{\Xi}. \tag{5d}$$

Note that ξ is used as an index for the second-stage decisions y in scenario ξ.

3.2. **Formal Statement of CCG.**

While Problem (5) is now a standalone optimization problem instead of a trilevel one, it typically contains a large number of variables and constraints. Moreover, the enumeration of points inside $\hat{\Xi}$ is, usually, impractical for real-world and large-scale applications. The key idea behind CCG is to initially remove variables and constraints in (5c) and (5d) from the problem and to iteratively generate them on the fly. Thus, at each iteration $t \in \mathbb{N}$, only a subset of scenarios $\Xi^t \subseteq \hat{\Xi}$ is considered and the following master problem is solved:

$$\begin{aligned} \min_{x,\theta} \inf_{y_\xi} \quad & c^\top x + \theta && \text{(6a)} \\ \text{s.t.} \quad & x \in X, && \text{(6b)} \\ & \theta \geq d(\xi)^\top y_\xi, \quad \xi \in \Xi^t, && \text{(6c)} \\ & y_\xi \in Y(x, \xi), \quad \xi \in \Xi^t. && \text{(6d)} \end{aligned}$$

Generating new variables and constraints from (5c) and (5d) is then done by solving the adversarial problem using the oracle in Assumption 4. A formal statement of the procedure is given in Algorithm 1.

Algorithm 1 Standard CCG method

```
 1: Input: an instance of (ARO) and an initial set $\Xi^0 \subset \hat{\Xi}$ such that (6) is bounded
    for $t = 0$.
 2: Set $\mathrm{LB} \leftarrow -\infty$, $\mathrm{UB} \leftarrow +\infty$, and $t \leftarrow 0$.
 3: while $\mathrm{UB} > \mathrm{LB}$ do
 4:     Solve the master problem (6).
 5:     if it is infeasible then
 6:         Return “Problem (ARO) is infeasible.”
 7:     end if
 8:     Let $v_{\mathrm{MP}}^t$ denote its objective value and $(x^t, \theta^t)$ its associated solution.
 9:     Set $\mathrm{LB} \leftarrow v_{\mathrm{MP}}^t$.
10:     Solve the adversarial problem (1) with $\hat{x} = x^t$.
11:     Let $v_{\mathrm{ADV}}^t$ be its objective value and $\xi^t$ its associated solution.
12:     Set $\mathrm{UB} \leftarrow \min\{\mathrm{UB}, c^\top x^t + v_{\mathrm{ADV}}^t\}$.
13:     Set $\Xi^{t+1} \leftarrow \hat{\Xi}^t \cup \{\xi^t\}$ and $t \leftarrow t + 1$.
14: end while
```

In Line 1, it is asked for an initial set of scenarios such that (6) is bounded. To obtain this, note that it is enough, as per Assumption 1, to solve the adversarial problem (1) for an arbitrary $\hat{x} \in X$. Indeed, if Problem (6) would be unbounded with such an initial scenario, this would mean that $\hat{x}$ is feasible and has an objective value of $-\infty$, which would contradict the assumption. Moreover, in a practical implementation, the stopping condition in Line 3 can also be checked after Line 9 so as to potentially terminate earlier.

The main argument for finite terminate is that, in the worst case, all scenarios in $\hat{\Xi}$ will be generated after $T := |\hat{\Xi}|$ iterations with $v_{\mathrm{MP}}^T = c^\top x^T + v_{\mathrm{ADV}}^T$, i.e., $\mathrm{UB} = \mathrm{LB}$. The interested reader is referred to the original papers by Zeng and Zhao (2013) and Zhao and Zeng (2012) for the details.

3.3. **Computational Challenges.** We now turn our attention to the computational behavior of CCG in case of mixed-integer second-stage variables. While in the continuous setting, generating new scenarios is not too harmful for the solution time of the master problem (as done at each iteration in Line 4 of Algorithm 1), it may have a dramatic impact in the mixed-integer case. Indeed, at each iteration, a new set of mixed-integer variables and constraints is added to the master problem, rapidly making it a large-scale MIP. Solving the lower-bounding problem (6) then becomes intractable quickly.

Indeed, plotting the empirical cumulative distribution function (ECDF) of computation times over a given test set of significant size, one can easily observe a similar curve as the one depicted in Figure 1. In this plot, one observes that instances

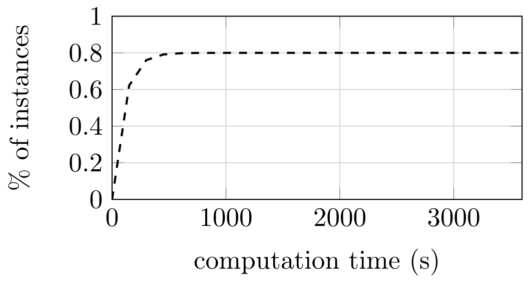

FIGURE 1. A typical ECDF of computation times of CCG over a fictitious set of test instances

that are solved within the time limit (here, fixed to 3600s for the example) are rapidly solved, while the other instances cannot be solved within the time limit. This behavior can be observed, e.g., in Lefebvre et al. (2023) on a facility location problem, and will be confirmed in Section 5. Roughly speaking, solved instances are those instances for which only a small number of iterations is needed to converge, while unsolved ones need a larger number of scenarios, and therefore end up with a larger optimization problem to solve.

To the best of our knowledge, this fact was first commented on by Subramanyam (2022), who wrote "the solution of [larger] instances is limited more by the lower bounding problem in the column-and-constraint generation algorithm, than by the computation of worst-case parameter realizations."

For simplicity, let us briefly assume that the second-stage's feasible space $Y(x,\xi)$ is described by a parameterized convex mixed-integer nonlinear program and let us consider a generic iteration $t \geq 0$. Then, solving Problem (6) is typically done by using branch-and-bound techniques based on continuous relaxations for obtaining valid lower bounds at each node. The issue with CCG for mixed-integer second-stage variables is that the root-node relaxation is obtained by dropping integrality requirements on both the here-and-now and wait-and-see decisions. This amounts to replace the nonconvex cuts

$$\theta \geq \inf_{y\in Y(x,\xi)} d(\xi)^\top y =: \mathcal{Q}(x,\xi),$$

which exactly reflect the objective function in (4), by the convex cuts

$$\theta \geq \inf_{y\in \operatorname{cont}(Y(x,\xi))} d(\xi)^\top y =: \bar{\mathcal{Q}}(x,\xi), \tag{7}$$

which do not completely reflect it. Indeed, it is well known that the continuous relaxation of a mixed-integer problem can be arbitrarily bad w.r.t. the original problem. While being computationally attractive (convex vs. nonconvex), the cuts in (7) have the clear disadvantage that, even if a feasible point $(\hat{x},\hat{\theta}) \in X \times \mathbb{R}$ is found during the exploration, we typically have

$$\hat{\theta} = \max_{\xi\in\Xi^t} \bar{\mathcal{Q}}(\hat{x},\xi) < \max_{\xi\in\Xi^t} \mathcal{Q}(\hat{x},\xi),$$

i.e., a feasible here-and-now decision does not necessarily lead to a valid upper bound of the problem. Thus, branching decisions in the x-space, which are driven by $\bar{\mathcal{Q}}$ instead of $\mathcal{Q}$, are based on a potentially wrong feedback regarding the second-stage problems. While these surrogate problems can be made tight by branching on second-stage decisions, it is likely to generate a large number of nodes.

Moreover, it is not trivial to decide which second-stage variable should be selected for branching. Indeed, because θ stands for the finite maximum of second-stage problems over the set of scenarios, many branching decisions will not have an impact on θ and, thus, on the lower bound. On the contrary, always branching on variables associated to the same scenario, say on an arbitrarily chosen $\hat{\xi}$, may lead to open nodes where the here-and-now decisions are such that $\operatorname{cont}(Y(x,\xi)) \neq \emptyset$ holds while $Y(x,\xi) = \emptyset$ holds for some $\xi \in \Xi^t \backslash \{\hat{\xi}\}$. Thus, one would dive in the direction of an infeasible sub-problem.

In the next section, using Assumptions 5 and 6, we introduce a new extended formulation, akin to (5), but having a much stronger relaxation, and which does not require branching on second-stage decisions. In particular, we will show that cuts defined in terms of $\mathcal{Q}$ are replaced, at the root node, by ones defined in terms of $\operatorname{vex}(\mathcal{Q})$.

4. A Stronger Formulation For Column-and-constraint Generation

We now state our main contribution, which is a new CCG algorithm. To this end, a new and stronger extended formulation of (ARO) is presented in Section 4.1. In Section 4.2, we show how its continuous relaxation can be solved using CG techniques and, in Section 4.3, we embed this relaxation in a branch-and-bound algorithm. A CG-based heuristic to find feasible points during the tree search is introduced in Section 4.4. Finally, the new CCG algorithm is formally stated in Section 4.5.

4.1. A New Extended Formulation.

Recall that the starting point of CCG is the equivalence between (ARO) and the extended formulation (4). By linearity of the objective function, it is also equivalent to replace the feasible space of the second-stage problem by its convex hull; see Tardella (2004). This leads to

$$\min_{x\in X} \left\{ c^\top x + \max_{\xi\in\hat{\Xi}} \inf_{y\in\operatorname{conv}(Y(x,\xi))} d(\xi)^\top y \right\}$$

with $\hat{\Xi} = \operatorname{vert}(\operatorname{conv}(\Xi))$ as before. With similar considerations as those in Section 3, we directly obtain the new extended formulation

$$\begin{aligned}
\min_{x,\theta} \inf_{y_\xi} \quad & c^\top x + \theta && \text{(8a)}\\
\text{s.t.} \quad & x \in X, && \text{(8b)}\\
& \theta \geq d(\xi)^\top y_\xi, \quad \xi \in \hat{\Xi}, && \text{(8c)}\\
& y_\xi \in \operatorname{conv}(Y(x,\xi)), \quad \xi \in \hat{\Xi}. && \text{(8d)}
\end{aligned}$$

The motivation for this formulation is that, in contrast to (5), second-stage variables y_ξ need not to take integer values. The drawback, however, lies in handling Constraint (8d). Such constraints require that y_ξ belongs to the convex hull of a decision-dependent set. While this is hard to deal with in general due to the inherent nonconvexity of (8d), a convex reformulation can be derived under Assumption 5. This has been done, e.g., by Arslan and Detienne (2022) and Detienne et al. (2024) for ARO problems with objective uncertainty. Later, it has also been extended in Lefebvre et al. (2023) for adjustable problems with discrete uncertainty sets. Based on these results, we now introduce our main theorem. This theorem is the foundation for the CCG algorithm introduced later.

Theorem 1. *Let $Y(\cdot,\cdot)$ as well $Z(\cdot)$ be defined as in Assumption 5 and let S be any subset of Ξ. Define $V_\xi := \operatorname{ext}(\operatorname{cl}(\operatorname{conv}(Z(\xi))))$ as well as $R_\xi := \operatorname{ray}(\operatorname{cl}(\operatorname{conv}(Z(\xi))))$ for each $\xi \in S$ and assume that at least one of the two following conditions hold:*

(i) For any point $\xi \in S$, the convex hull of $Z(\xi)$ is closed.

(ii) For any $\hat{x} \in X$ and any $\hat{\xi} \in S$, the relative interior of $Y(\hat{x},\hat{\xi})$ is non-empty.

Then, the optimization problem

$$\begin{aligned}
\min_{x,\theta,y_\xi} \quad & c^\top x + \theta && \text{(9a)}\\
\textit{s.t.} \quad & x \in X, && \text{(9b)}\\
& \theta \geq d(\xi)^\top y_\xi, \quad \xi \in S, && \text{(9c)}\\
& y_\xi \in \operatorname{cl}(\operatorname{conv}(Y(x,\xi))), \quad \xi \in S, && \text{(9d)}
\end{aligned}$$

is equivalent to

$$\begin{aligned}
\min_{x,\theta} \inf_{\alpha_\xi(\cdot),\beta_\xi(\cdot)} \quad & c^\top x + \theta && \text{(10a)}\\
\textit{s.t.} \quad & x \in X, && \text{(10b)}\\
& \theta \geq d(\xi)^\top \left(\int_{V_\xi} y \, \mathrm{d}\alpha_\xi(y) + \int_{R_\xi} y \, \mathrm{d}\beta_\xi(y) \right), \quad \xi \in S, && \text{(10c)}\\
& \int_{V_\xi} y_i \, \mathrm{d}\alpha_\xi(y) \leq u_i(1 - x_i), \quad i \in I, \ \xi \in S, && \text{(10d)}\\
& \int_{V_\xi} \mathrm{d}\alpha_\xi(y) = 1, \qquad \xi \in S, && \text{(10e)}\\
& \alpha_\xi \in M(V_\xi), \quad \xi \in S, && \text{(10f)}\\
& \beta_\xi \in M(R_\xi) \quad \xi \in S, && \text{(10g)}
\end{aligned}$$

in the sense that any feasible point of the one can be mapped to a feasible point of the other with the same objective function value. Moreover, if Condition (i) holds, then

$$\operatorname{cl}(\operatorname{conv}\left(Y(\hat{x},\hat{\xi})\right)) = \operatorname{conv}\left(Y(\hat{x},\hat{\xi})\right)$$

holds for any $\hat{\xi} \in \hat{\Xi}$ and any $\hat{x} \in X$.

Before giving a proof of Theorem 1, let us first discuss its interpretation. As a first remark, we would like to emphasize the difference between the extended formulation (8) and Model (9). In the first, we consider the convex hull of $Y(x,\xi)$ for all $\xi \in \Xi$ and $x \in X$, while we consider its closure in the second. Thus, Theorem 1 treats a relaxation of (8) under the special case in which $S = \hat{\Xi}$ since $\operatorname{conv}(Y(x,\xi)) \subseteq \operatorname{cl}(\operatorname{conv}(Y(x,\xi)))$. This is further exploited later in this paper.

To ease our discussion, let us assume that the continuous relaxation of X is convex. Note that Constraints (9d) are nonconvex since both y_ξ and x are decision variables. For a complete discussion on this type of constraints, we refer to Arslan and Detienne (2022); in particular to Figure 2 and the related comments. Thus, Theorem 1 states the equivalence between (9), whose continuous relaxation is a nonconvex optimization problem, and another optimization problem, whose continuous relaxation is convex. This is computationally attractive. However, this is obtained by moving from the vector space $\mathbb{R}^{n_y}$ to the space of measures $M(V_\xi)$ and $M(R_\xi)$ defined on V_ξ and R_ξ. Yet, a direct connection between these spaces can be found. To this end, let $\xi \in S$ be fixed, and consider a measure $\alpha_\xi \in M(V_\xi)$ satisfying (10e). Then, the integral $\int_{V_\xi} y \, \mathrm{d}\alpha_\xi(y)$ simply denotes a convex combination of points in V_ξ and thus belongs to $\mathbb{R}^{n_y}$. Similarly, consider $\beta_\xi \in M(R_\xi)$. Then, $\int_{R_\xi} y \, \mathrm{d}\beta_\xi(y)$ is a nonnegative sum of points in R_ξ. Indeed, one easily shows that

$$\int_{V_\xi} y \, \mathrm{d}\alpha_\xi(y) + \int_{R_\xi} y \, \mathrm{d}\beta_\xi(y) \in \operatorname{conv}(V_\xi) + \operatorname{cone}(R_\xi) \subseteq \mathbb{R}^{n_y} \qquad (11)$$

holds for any feasible α_ξ and β_ξ in (10). Thus, (11) can be understood as a Dantzig–Wolfe reformulation of $\operatorname{cl}(\operatorname{conv}(Z(\xi)))$, i.e., it is expressed as the set of convex combinations of its extreme points and nonnegative sum of its extreme rays. The use of Lebesgue–Stieltjes measures allows us to consider cases in which V_ξ and R_ξ are potentially uncountable but reduces to the classic result if V_ξ and R_ξ are finite.

All in all, Theorem 1 claims the equivalence between (9) and (10), where the last model is derived thanks to a Dantzig–Wolfe reformulation of the closure of the convex hull of $Z(\xi)$. We now prove Theorem 1.

Proof of Theorem 1. For the ease of presentation, let us first define the set-valued map L via $L(x) := \{y \in \mathbb{R}^{n_y} : 0 \leq y_i \leq u_i(1 - x_i),\ i \in I\}$, i.e., the set of points that satisfy the linking constraints. Recall that, by Assumption 5, we have

$$Y(x, \xi) = Z(\xi) \cap L(x).$$

Thus, by Theorem A.1 in Lefebvre et al. (2023), it holds[1]

$$\operatorname{conv}(Y(x, \xi)) = \operatorname{conv}(Z(\xi)) \cap L(x),$$

for any $x \in X$ and $\xi \in \Xi$. We then have two cases. First, assume that Condition (i) is satisfied. By the latter, it holds that $\operatorname{cl}(\operatorname{conv}(Z(\xi))) = \operatorname{conv}(Z(\xi))$, showing that $\operatorname{conv}(Y(x, \xi))$ is closed due to Assumption 5, and thus, equals its closure. Second, assume that the Condition (ii) is satisfied. Then, $Y(x, \xi) \neq \emptyset$ implies that $Z(\xi)$ and $L(x)$ have at least one point in common. By Theorem 6.5 of Rockafellar (1970), it then holds

$$\operatorname{cl}(\operatorname{conv}(Y(x, \xi))) = \operatorname{cl}(\operatorname{conv}(Z(\xi)) \cap L(x)) = \operatorname{cl}(\operatorname{conv}(Z(\xi))) \cap L(x).$$

The final result is achieved by using an internal representation of $\operatorname{cl}(\operatorname{conv}(Z(\xi)))$, i.e., a Dantzig–Wolfe reformulation, expressing it as the set of convex combinations of extreme points of V_ξ and the nonnegative sum of extreme rays in R_ξ. Indeed, by Theorem 18.5 of Rockafellar (1970), we have

$$\operatorname{cl}(\operatorname{conv}(Z(\xi))) = \operatorname{conv}(V_\xi) + \operatorname{cone}(R_\xi).$$

Note that, since y_I is upper bounded by u_I in $Z(\xi)$, any $y \in R_\xi$ satisfies $y_I = 0$. The claimed result is obtained by expressing $\operatorname{conv}(V_\xi)$ and $\operatorname{cone}(R_\xi)$ through measures defined over V_ξ and R_ξ. Indeed, we have that y is in the convex hull of V_ξ if and only if there exists a probability measure α_ξ defined over V_ξ such that (see § 2.1.4 in Boyd and Vandenberghe (2004))

$$y = \int_{V_\xi} y' \, \mathrm{d}\alpha_\xi(y').$$

Similar arguments can be used for $\operatorname{cone}(R_\xi)$. □

The extended models (9) and (10) can be seen as a relaxation of (8), and, thus, of (ARO). Indeed, in general, it may be that the inclusion $S \subseteq \hat{\Xi}$ is strict or that $\operatorname{conv}(Y(x, \xi)) \subset \operatorname{cl}(\operatorname{conv}(Y(x, \xi)))$ holds. In the next proposition, we give a condition under which (9), (10), and (ARO) are equivalent.

Proposition 2. *Let Assumptions 2–6 hold, and assume that points in S have been generated by the oracle in Assumption 4. Then, Condition (i) of Theorem 1 holds. If, moreover, $S = \hat{\Xi}$, then (9), (10), and (ARO) are equivalent in the sense that they have the same optimal objective value and the same set of here-and-now solutions.*

[1]Note that Theorem A.1 in Lefebvre et al. (2023) requires the boundedness of $Y(x, \xi)$ but can be easily adapted to the case in which only interdicted variables need to be bounded, which is the case in our setup.

Proof. By assumption, any point in S has been generated by the oracle in Assumption 4. Thus, by Assumption 6, the convex hull of $Z(\xi)$ is closed. This proves the first statement. Note that, by Theorem 1, we already have that (9) and (10) are equivalent and that the convex hull of $Y(\hat{x}, \hat{\xi})$ is closed for any $\hat{x} \in X$ and any $\hat{\xi} \in S$. It is now sufficient to show that (9) and (ARO) are equivalent if S contains all the extreme points of the convex hull of Ξ. This is implied by the quasi-convexity of $\mathcal{Q}$ (see Assumption 3), and the polyhedral nature of $\operatorname{conv}(\Xi)$; see Assumption 2. □

Consequently, Problem (10) can be used in place of (6) in the standard CCG so as to obtain an exact algorithm for Problem (ARO); see Algorithm 1. However, effectively solving (10) still poses some computational challenges. Indeed, it is a large-scale and potentially semi-infinite optimization problem—even if S is finite. This will be treated in the next section.

4.2. Solving the Master Problem.

In this section, we show how Problem (10) can be solved (possibly to ε-optimality) by using CG techniques. More precisely, we derive a CG algorithm for solving its continuous relaxation. In the next section, we discuss how this can be embedded in a branch-and-bound algorithm so as to enforce integrality requirements. For simplicity, we assume that S is finite and that the continuous relaxation of X is given by

$$\operatorname{cont}(X) = \{x \in \mathbb{R}^{n_x} : Ax \geq b\}.$$

The key idea behind CG is to replace the sets V_ξ and R_ξ by proper finite subsets $\hat{V}_\xi$ and $\hat{R}_\xi$. By doing so, one obtains a so-called restricted master problem (RMP) with a reasonable and, most importantly, finite number of columns. Thus, the RMP leads to an upper bound on the original full problem. In our case, the RMP reads

$$\min_{x,\theta,\alpha_\xi,\beta_\xi} \quad c^\top x + \theta \tag{RMP.a}$$

$$\text{s.t.} \quad Ax \geq b, \tag{RMP.b}$$

$$\theta \geq d(\xi)^\top \left(\sum_{v \in \hat{V}_\xi} \alpha_{\xi,v} v + \sum_{r \in \hat{R}_\xi} \beta_{\xi,r} r \right), \quad \xi \in S, \tag{RMP.c}$$

$$\sum_{v \in \hat{V}_\xi} \alpha_{\xi,v} v_i \leq u_i(1 - x_i), \quad i \in I, \ \xi \in S, \tag{RMP.d}$$

$$\sum_{v \in \hat{V}_\xi} \alpha_{\xi,v} = 1, \quad \xi \in S, \tag{RMP.e}$$

$$\alpha_{\xi,v} \geq 0, \quad v \in \hat{V}_\xi, \ \xi \in S, \tag{RMP.f}$$

$$\beta_{\xi,r} \geq 0, \quad r \in \hat{R}_\xi, \ \xi \in S. \tag{RMP.g}$$

Then, new points in $\hat{V}_\xi$ and $\hat{R}_\xi$ are sequentially generated so as to decrease the upper bound until some stopping criterion is satisfied.

Following the standard CG approach (Desrosiers and Lübbecke 2005), one considers the dual of (RMP) to derive the pricing problem (PP) to identify missing columns, i.e., points in V_ξ and R_ξ in the RMP. The dual problem is given as

$$\begin{aligned}
\max_{\gamma,\mu_\xi,\lambda_\xi,\pi_\xi} \quad & b^\top\gamma + \sum_{\xi\in S}\left(\pi_\xi + u_I^\top\lambda_\xi\right) && \text{(13a)}\\
\text{s.t.} \quad & \sum_{\xi\in S}\mu_\xi = 1, && \text{(13b)}\\
& A^\top\gamma - \sum_{\xi\in S}\begin{pmatrix}0\\ u_I\circ\lambda_\xi\end{pmatrix} = c, && \text{(13c)}\\
& \gamma \geq 0, && \text{(13d)}\\
& \pi_\xi \leq d(\xi)^\top v\mu_\xi - v_I^\top\lambda_\xi, \quad v\in\hat V_\xi,\ \xi\in S, && \text{(13e)}\\
& 0 \leq d(\xi)^\top r\mu_\xi, \quad r\in\hat R_\xi,\ \xi\in S, && \text{(13f)}\\
& \mu_\xi \geq 0, \quad \xi\in S, && \text{(13g)}\\
& \lambda_\xi \leq 0, \quad \xi\in S. && \text{(13h)}
\end{aligned}$$

Here, "$\circ$" denotes the Hadamard product of two vectors and we assume that the variables indexed by I are the last ones in the variable vector x. Constraints (13b) and (13c) are the dual constraints associated to the here-and-now variables θ and x, respectively, while the constraints in (13e) and (13f) are associated to second-stage decisions $\alpha_{\xi,v}$ and $\beta_{\xi,r}$. Then, identifying missing columns amounts to identifying missing dual constraints, which can be done by separation. We now discuss how points in V_ξ and R_ξ are generated.

4.2.1. *Generating Extreme Points from V_ξ.* Consider, for now, that $\hat R_\xi = R_\xi$, i.e., all extreme rays of the closure of the convex hull of $Z(\xi)$ are present in the RMP. Thus, that only extreme points in V_ξ need to be generated. This reduces to identify a dual constraint

$$\pi_\xi \leq d(\xi)^\top v\mu_\xi - v_I^\top\lambda_\xi$$

for some $\xi\in S$, which is missing in the dual problem (13), i.e., to find $v\in V_\xi\backslash\hat V_\xi$. To this end, let $(\gamma^*,\mu_\xi^*,\lambda_\xi^*,\pi_\xi^*)$ be a given solution of (13). Pricing requires to solve, for each $\xi\in S$,

$$\rho_\xi^* := \min_{y\in V_\xi}\rho(y;\mu_\xi^*,\lambda_\xi^*,\pi_\xi^*) := d(\xi)^\top y\mu_\xi^* - y_I^\top\lambda_\xi^* - \pi_\xi^*,$$

which is called the reduced cost of y. If, for all $\xi\in S$, ρ_ξ^* is nonnegative, then all points in V_ξ, which are necessary for (10) to fully represent (9), are already present in $\hat V_\xi$ and the algorithm can stop—though, most typically, $\operatorname{conv}(\hat V_\xi) = \operatorname{cl}(\operatorname{conv}(Z(\xi)))$ does not hold. Otherwise, there exists a scenario $\hat\xi\in S$ with $\rho_{\hat\xi}^* < 0$, and the point $y_{\hat\xi}^*$ realizing the minimum represents a violated constraint. Hence, it must be added to $\hat V_\xi$. In the primal, this amounts to a new column being generated, associated to a variable $\alpha_{\hat\xi,y_{\hat\xi}^*}\geq 0$.

Recall that V_ξ denotes the set of extreme points of the closure of the convex hull of $Z(\xi)$. Thus, by linearity of the objective function, pricing a new column (for $\xi\in S$) reduces to solving

$$\inf_{y\in Z(\xi)}\rho(y;\mu_\xi^*,\lambda_\xi^*,\pi_\xi^*). \tag{14}$$

Thus, pricing a new column can be done by solving $|S|$ optimization problems over $Z(\xi)$, which is a non-interdicted version of the second-stage problem, under scenario ξ. We would like to point out that these problems can be solved in parallel to speed-up the overall scheme.

4.2.2. *Generating Extreme Rays from R_ξ.* Dropping the assumption that $\hat{R}_\xi = R_\xi$ holds possibly leads to situations in which the pricing problem (14) is unbounded, i.e., $\rho^*_\xi = -\infty$ holds for some $\xi \in \Xi$. In this case, there exists an extreme ray y^* of $\operatorname{cl}(\operatorname{conv}(Z(\xi)))$ pointing in the direction of unboundedness. A new constraint of the form

$$0 \leq d(\xi)^\top y^* \mu_\xi$$

is then added to $\hat{R}_\xi$. In the primal, a column associated to a variable $\beta_{\xi,y^*} \geq 0$ is generated.

4.2.3. *Computing Lower Bounds.* Whenever a pricing operation leads to the generation of a point in V_ξ, a lower bound on the value of (9) can be computed. Indeed, it always holds

$$v^*_{\text{RMP}} + \sum_{\xi \in \hat{\Xi}} \rho^*_\xi \leq \bar{v}^* \leq v^*_{\text{RMP}}, \tag{15}$$

where v^*_{RMP} denotes the current value of the RMP and $\bar{v}^*$ denotes the optimal objective value of the continuous relaxation of (9), obtained by replacing X by $\operatorname{cont}(X)$. The interested reader is referred to Desrosiers and Lübbecke (2005) for more details. This result can be used to obtain an ε-optimal solution for the continuous relaxation of (10).

The complete CG algorithm is stated in Algorithm 2. In our implementation, we use[2]

$$\operatorname{gap}(\text{LB}, \text{UB}) := \frac{\text{UB} - \text{LB}}{10^{-10} + |\text{UB}|}.$$

Proofs of convergence of the algorithm can be found in Wolsey (1998) for cases in which $\operatorname{cont}(X)$ is a polyhedron and in García et al. (2003) for more general cases.

Algorithm 2 Column Generation for (10)

```
1: **Input**: $\varepsilon \geq 0$ and an initial set of points $\hat{V}_\xi$ and $\hat{R}_\xi$ such that (RMP) is feasible.
2: Set LB $\leftarrow -\infty$ and UB $\leftarrow +\infty$.
3: **while** $\operatorname{gap}(\text{LB}, \text{UB}) > \varepsilon$ **do**
4:     Solve (RMP).
5:     Let $v^*_{\text{RMP}}$ denote its objective value and $(\gamma^*, \mu^*_\xi, \lambda^*_\xi, \pi^*_\xi)$ its dual solution.
6:     Set UB $\leftarrow v^*_{\text{RMP}}$.
7:     **for** $\xi \in \hat{\Xi}$ **do**
8:         Solve the pricing problem (14) associated to $\xi$.
9:         Let $\rho^*_\xi$ denote its objective value.
10:        **if** $\rho^*_\xi > -\infty$ **then**
11:            Let $v$ be a point realizing the infimum of (14).
12:            Set $\hat{V}_\xi \leftarrow \hat{V}_\xi \cup \{v\}$.
13:        **else**
14:            Let $r$ be a direction of unboundedness of (14).
15:            Set $\hat{R}_\xi \leftarrow \hat{R}_\xi \cup \{r\}$.
16:        **end if**
17:    **end for**
18:    Set $\text{LB}_{\text{iter}} \leftarrow v^*_{\text{RMP}} + \sum_{\xi \in \hat{\Xi}} \rho_\xi$.
19:    Set LB $\leftarrow \max\{\text{LB}, \text{LB}_{\text{iter}}\}$.
20: **end while**
```

[2]This is motivated by
https://www.ibm.com/docs/en/icos/22.1.1?topic=parameters-relative-mip-gap-tolerance.

Remark 3. *Algorithm 2 requires initial sets $\hat{V}_\xi$ and $\hat{R}_\xi$ such that* (RMP) *is feasible. Note that it is always possible to find such initial sets by working on a feasibility variant of* (RMP) *using slack variables. This is similar to Phase-I of the simplex method to find a first basic solution of an LP. In some more specific cases, e.g., when the RMP is formulated as a conic program, Farkas pricing can also be used; see, e.g., Achterberg (2009). Note that both approaches can be used to detect infeasible problems.*

Remark 4. *By Assumption 1 and Equation* (15)*, the RMP is always bounded from below.*

4.3. **Branch-and-Price.** In the previous section, we discussed how the continuous relaxation of (10) can be solved by CG. By doing so, one obtains a solution $(x^*, \theta^*, \alpha^*_{\xi,v}, \beta^*_{\xi,r})$ for which $x^* \in \operatorname{cont}(X)$ holds. If $x^* \in X$ holds, then x^* solves (9). Otherwise, x^* is not a feasible point of (9) for two reasons: (i) some integrality requirements of X are not fulfilled or (ii) the inclusion $\operatorname{conv}(Y(x,\xi)) \subseteq Z(\xi) \cap L(x)$ is strict. However, we have that the value, i.e., $c^\top x^* + \theta^*$, is a valid lower bound on v^*.

These considerations are the key ingredients for embedding the lower bounding problem (9) in a branch-and-bound algorithm. We start by solving the root-node problem in which all integrality requirements are dropped. If the problem is infeasible, the algorithm stops, and (9), thus (ARO), is infeasible. Otherwise, we let $(x^0, \theta^0, \alpha^0_\xi, \mu^0_\xi)$ denote its solution. If $x^0 \in X$, then the algorithm stops, and x^0 solves (9) with $v^* = c^\top x^0 + \theta^0$. Otherwise, we perform branching. To this end, we let $L \leftarrow \{0\}$ be the set of active nodes, and set the best known lower bound LB to $c^\top x^0 + \theta^0$. The best upper bound is initially set to ∞.

Then, a node q is selected and removed from L. A variable index $j \in \{1, \dots, n_x\}$ with $x^q_j \notin \mathbb{Z}$ is chosen, giving rise to two new nodes. These new nodes, noted ℓ and r, are such that $x \le \lfloor x^q \rfloor$ in ℓ and $x \ge \lceil x^q \rceil$ in r are added. We solve both nodes and check for feasibility. If a node is infeasible, it is discarded. If $x^\ell \in X$ (or $x^r \in X$), a new feasible point has been found and we let $\text{UB} \leftarrow \min\{\text{UB}, c^\top x^\ell + \theta^\ell\}$, or $c^\top x^r + \theta^r$. Otherwise, we add them to L. The best lower bound can be updated using the node q' with $q' \in \arg\min\{c^\top x^k + \theta^k : k \in L\}$. If no such node exists, i.e., $L = \emptyset$, the algorithm stops, and the incumbent solution solves (9). If no incumbent has been found, the problem is infeasible. The algorithm can also be stopped whenever $\operatorname{gap}(\text{LB}, \text{UB}) \le \varepsilon$ for some $\varepsilon \ge 0$.

It is important to highlight that branching is only performed on the x variables and not on y_ξ variables. This is in contrast to classic applications of Dantzig–Wolfe reformulations (or discretization approaches) used for solving large-scale MIP. The reason for this is the specific nature of the reformulation stated in Theorem 1.

4.4. **A CG-Based Heuristic.** Empirical evidence shows that the performance of branch-and-bound algorithms highly depends on the availability of good lower bounds at the root node and on the early detection of good feasible points. While improving the quality of the lower bound (compared to the classic CCG) is the main motivation behind Theorem 1, we derive as a by-product of this theorem a simple heuristic to compute good feasible points throughout the algorithm.

Consider a node q such that some integer variables x_j have fractional value and let $(\hat{V}^q_\xi)_S$ and $(\hat{R}^q_\xi)_S$ be the set of generated columns in node q. The following model can be used to obtain a feasible point of (9) or (10):

$$\min_{x,\theta,\alpha_{\xi,v},\beta_{\xi,r}} \quad c^\top x + \theta \tag{16a}$$

$$\text{s.t.} \quad x \in X, \tag{16b}$$

$$\theta \geq d(\xi)^\top \left(\sum_{v \in \hat{V}^q_\xi} \alpha_{\xi,v} v + \sum_{r \in \hat{R}^q_\xi} \beta_{\xi,r} r \right), \quad \xi \in S, \tag{16c}$$

$$\sum_{v \in \hat{V}^q_\xi} \alpha_{\xi,v} v_i \leq u_i(1 - x_i), \quad i \in I, \ \xi \in S, \tag{16d}$$

$$\sum_{v \in \hat{V}^q_\xi} \alpha_{\xi,v} = 1, \quad \xi \in S, \tag{16e}$$

$$\alpha_{\xi,v} \geq 0, \quad v \in \hat{V}^q_\xi, \ \xi \in S, \tag{16f}$$

$$\beta_{\xi,r} \geq 0, \quad r \in \hat{R}^q_\xi, \ \xi \in S. \tag{16g}$$

In this model, the generated columns are kept and used to approximate $\operatorname{conv}(Z(\xi))$. Moreover, integrality requirements are applied, i.e., we have $x \in X$ instead of $x \in \operatorname{cont}(X)$. This model is then solved as a MIP. Assume that it is feasible and let $(x^{\mathrm{H}}, \theta^{\mathrm{H}}, \alpha^{\mathrm{H}}_{\xi,v}, \mu^{\mathrm{H}}_{\xi,r})$ be its solution. Clearly, x^{H} is feasible for (10). However, we typically have

$$\theta^{\mathrm{H}} > \max_{\xi \in S} \min_{y \in Y(x^{\mathrm{H}},\xi)} d(\xi)^\top y.$$

If the resulting problem is infeasible, no feasible point is computed, and the heuristic stops.

4.5. Formal Statement of the Procedure.

In this section, we formally state the CCG algorithm enhanced with CG. Its correctness directly follows from Theorem 1 and from the closedness of $\operatorname{conv}(Z(\xi))$ for all $\xi \in \hat{\Xi}$ generated by the adversarial problem. Finite termination is then implied by the finite termination of the CG procedure and the finite size of the branch-and-bound tree. The proposed scheme is formally stated in Algorithm 3.

Algorithm 3 CCG+CG

```
1: **Input**: an instance of (ARO) and initial set $\Xi^0 \subseteq \hat{\Xi}$ such that (9) is bounded if
   $S = \Xi^0$.
2: Set LB $\leftarrow -\infty$, UB $\leftarrow +\infty$, and $t = 0$.
3: while UB > LB do
4:     Solve the master problem (9) with $S = \Xi^t$ using a B&P algorithm.
5:     if it is infeasible then
6:         Return "Problem (ARO) is infeasible."
7:     end if
8:     Let $v^t_{\mathrm{MP}}$ denote its objective value and $(x^t, \theta^t)$ its associated solution.
9:     Set LB $\leftarrow v^t_{\mathrm{MP}}$.
10:    Solve the adversarial problem (1) with $\hat{x} = x^t$.
11:    Let $v^t_{\mathrm{ADV}}$ be its objective value and $\xi^t$ its associated solution.
12:    Set UB $\leftarrow \min\{\mathrm{UB}, c^\top x^t + v^t_{\mathrm{ADV}}\}$.
13:    Set $\Xi^{t+1} \leftarrow \hat{\Xi}^t \cup \{\xi^t\}$ and $t \leftarrow t + 1$.
14: end while
```

Theorem 3. *Suppose that Assumptions 1–6 hold. Then, Algorithm 3 terminates after a finite number of iterations T with x^T solving* (ARO) *and $v^* = \mathrm{UB}$.*

5. Computational Results

We now apply the CCG algorithm to three different applications. The first one is a facility location problem with facility disruption, the second one is a generalized assignment problem introduced by Bodur et al. (2024), and the third one is a scheduling problem, which is a variant of the $1|r_j|\sum w_j U_j$ scheduling problem; see Pinedo (2008). This means that the latter problem is about minimizing the weighted number of tardy jobs and the job processing times are uncertain.

5.1. **Experimental Setting.** To assess the computational benefits of the method, we compare it with the standard CCG approach as described by Zeng and Zhao (2013) and Zhao and Zeng (2012). The latter approach is denoted by CCG+Gurobi in this section. Additionally, we denote by CCG+CG the algorithm introduced in this paper in which the master problem is solved by branch-and-price without the CG-based heuristic, i.e., the heuristic presented in Section 4.4 is not used. The overall algorithm obtained by additionally using the CG-based heuristic is denoted by CCG+CG+H.

5.2. **Implementation.** The methods are implemented in C++17 using Gurobi (Gurobi Optimization, LLC 2023) version 11.0.0 for solving all appearing optimization problems in both methods, i.e., the master problem and the adversarial problem in standard CCG as well as the restricted master problem and the pricing problem in CG. The branch-and-price algorithm is implemented using the open-source library idol (Lefebvre 2023). Our code and instances are publicly available online at `https://github.com/hlefebvr/AE-using-column-generation-in-column-and-constraint-generation`.

At each node of the branch-and-price tree, CG is stabilized using dual-price smoothing as described by Pessoa et al. (2013). During each pricing step, for each scenario, a maximum of 20 columns are added to the restricted master problem. Only columns with negative reduced costs are added. The pricing problems are always solved to global optimality using Gurobi. Whenever the restricted master problem is infeasible, Farkas pricing is used to identify missing columns or to prove infeasibility of the node. When applicable, and for feasible nodes only, the heuristic introduced in this paper is called at every node of the branch-and-price tree whenever a point violating integrality requirements is found. Nodes are selected according to the "best bound first" rule, i.e., a node with the largest lower bound is selected. Variables are selected according to the "most infeasible" rule, i.e., a variable with fractional part closest to 0.5 is selected. No sub-tree exploration is performed.

In all three applications, the separation problem is solved by a branch-and-cut algorithm which makes use of interdiction cuts as described in Fischetti et al. (2019). This method was implemented in Gurobi using lazy constraint callbacks.

All experiments were conducted on a single core Intel Xeon Gold 6126 at 2.6 GHz with 16 GB of RAM. We use the default settings of Gurobi except for the maximum number of threads, which is set to 1, i.e., the parameter Threads is set to 1. Hence, all underlying optimization problems are solved with an optimality gap of 10^{-4}. We used the same tolerance for solving the master problem by column generation.

5.3. **Facility Location Problem with Facility Disruption.**

5.3.1. *Problem Statement.* We consider the robust facility location problem (FLP) as introduced by Lefebvre et al. (2023). We let V_1 be the set of candidates for opening facilities and let V_2 be the set of customers to be served. Each connection between a site $i \in V_1$ and a customer $j \in V_1$ is associated to a transportation cost denoted by $c_{ij} > 0$. For each site $i \in V_1$, we let its maximum capacity be $q_i > 0$ and its fixed cost for opening a facility be $f_i > 0$. Each client $j \in V_2$ is associated to a demand d_j and a profit p_j, which is earned if and only if the customer is served by a facility. Customers must be served by a unique facility, i.e., no demand split is possible.

In this application, facilities are subject to random disruptions making them unusable for delivery. We let $\Xi \subseteq \{0,1\}^{|V_1|}$ be the set of all possible disruptions. In line with the Γ-robust approach, we account for the fact that, most likely, not all facilities will be disrupted at the same time by limiting the maximum number of disrupted facilities to a fixed value $\Gamma \in \mathbb{N}$ with $\Gamma \leq |V_1|$; see Bertsimas and Sim (2004).

Now, let $\xi \in \Xi$ be a given disruption scenario with the interpretation that $\xi_i = 1$ if and only if facility $i \in V_1$ is disrupted. The deterministic FLP then reads

$$\begin{aligned}
\min_{x,y} \quad & \sum_{i \in V_1} f_i x_i + \sum_{i \in V_1} \sum_{j \in V_2} (c_{ij} - p_j) y_{ij} && \text{(17a)}\\
\text{s.t.} \quad & \sum_{i \in V_1} y_{ij} \leq 1, \quad j \in V_2, && \text{(17b)}\\
& \sum_{j \in V_2} d_j y_{ij} \leq q_i x_i (1 - \xi_i), \quad i \in V_1, && \text{(17c)}\\
& y_{ij} \in \{0,1\}, \quad i \in V_1, \ j \in V_2, && \text{(17d)}\\
& x_i \in \{0,1\}, \quad i \in V_1. && \text{(17e)}
\end{aligned}$$

Here, variables x are to be interpreted such that, for a location $i \in V_1$, $x_i = 1$ holds if and only if a facility i is opened. For each location $i \in V_1$ and each customer $j \in V_2$, y_{ij} is a binary variable indicating if facility i serves customer j. Thus, the constraints in (17b) ensure that each customer is served by at most one facility. The constraints in (17c) make sure that the capacity of each facility is not exceeded. Moreover, they link the opening and routing decisions and account for potential disruptions of facilities.

We derive the ARO version of this problem by letting the set of here-and-now decisions be the opening decisions, i.e., we set $X = \{0,1\}^{|V_1|}$, and define the set of feasible wait-and-see decisions as

$$Y(x,\xi) = \left\{ y \in \{0,1\}^{|V_1| \times |V_2|} : \text{(17b)–(17c)} \right\},$$

where $x \in X$ is a fixed here-and-now decision and $\xi \in \Xi$ is a given disruption scenario. Thus, the ARO problem considered is given by

$$\min_{x \in X} \left\{ \sum_{i \in V_1} f_i x_i + \max_{\xi \in \Xi} \min_{y \in Y(x,\xi)} \sum_{i \in V_1} \sum_{j \in V_2} (c_{ij} - p_j) y_{ij} \right\}.$$

5.3.2. *Instances.* The instances are randomly generated according to Lefebvre et al. (2023). The number of facility locations and the number of customers are taken as input. Then, for each facility location $i \in V_1$, the capacity q_i is drawn uniformly between 10 and 160. The opening cost f_i is computed as $f_i = \alpha_i + \beta_i \sqrt{q_i}$, where $\alpha_i \in [0, 90]$ and $\beta_i \in [100, 110]$ are randomly generated numbers. Demands are uniformly generated and scaled so that $\sum_{i \in V_1} q_i / \sum_{j \in V_2} d_j = \mu$ holds with μ being a given parameter. To compute transportation costs, a point in the unit square is randomly assigned to each facility location and each customer. Then, each pair $(i,j) \in V_1 \times V_2$ is associated to a transportation cost t_{ij} equal to ten times the Euclidean distance between i and j. The profit for serving a customer $j \in V_2$ is set to $p_j = 4 \times \text{median}\{t_{ij} : i \in V_1\}$.

For our test set, we choose $\mu \in \{2, 3\}$, the number of facility locations is taken in $\{10, 15\}$, whereas the number of customers take the value 20, 30, or 40. For each combination of these parameters, 5 instances are generated, producing a set

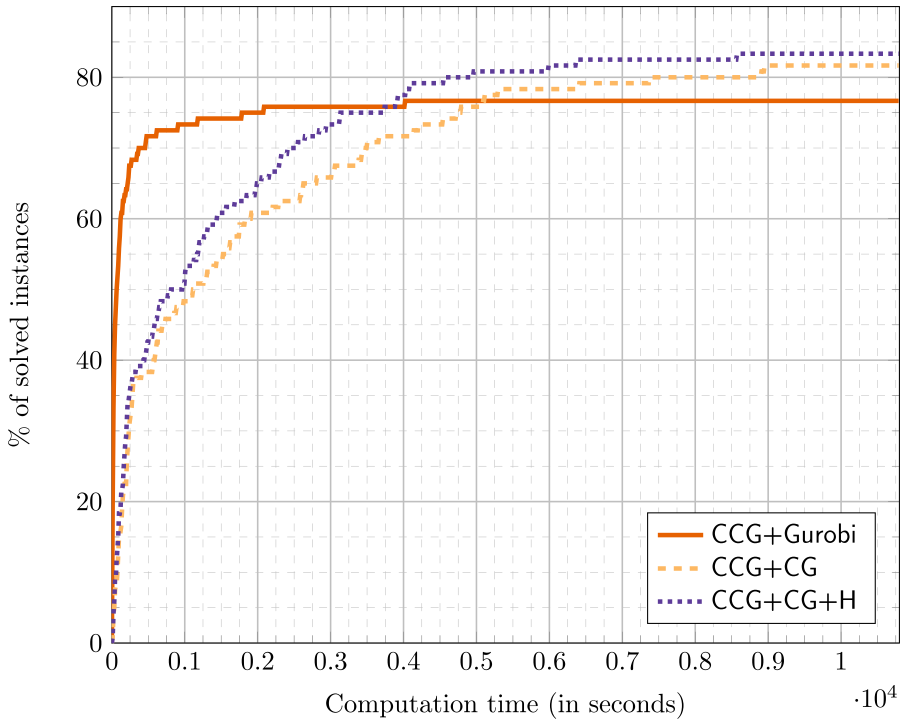

FIGURE 2. ECDFs of computation times for the FLP

of 60 nominal instances. Each instance is then solved by considering a disruption budget $\Gamma \in \{3, 4\}$ leading to 120 robust instances.

5.3.3. *Discussion of the Results.* For a time limit of 3 hours, Figure 2 depicts the ECDF of computation times over our test set, for all three methods. First, it can be seen that using CCG+Gurobi is beneficial for those instances that can be solved in less than 1 hour. However, CCG+Gurobi is not able to solve harder instances within the time limit. This is in line with the observation made in Section 3 and, in particular, Figure 1. In contrast, it can be seen that using CCG+CG and CCG+CG+H allows to solve more instances. Indeed, while the standard CCG+Gurobi approach solves around 75 % of the instances within the time limit, CCG+CG is able to solve more than 80 %. Moreover, the impact of the CG-based heuristic developed in Section 4.4 is clearly shown since it outperforms both CCG+Gurobi and CCG+CG for those instances which require more than 1 hour to be solved.

Figure 3 depicts the ECDF of memory used in percentage w.r.t. the maximum memory available of 16 GB. Clearly, both CCG+CG and CCG+CG+H outperform the standard version CCG+Gurobi. This can be explained by smaller sizes of the search trees when solving the master problems during the execution of the CCG algorithm. Hence, this shows that the quality of the dual bound at every iteration is stronger when column generation is used instead of simply Gurobi. However, solving each node is more demanding in terms of computation time.

More detailed statistics are reported in Table 1 and Table 2 depending on how the master problem is solved during the CCG, i.e., by CCG+Gurobi, CCG+CG, or CCG+CG+H. Table 1 regards those instances which could be solved within the time limit. In particular, "$|V_1|$" is the number of facilities, "$|V_2|$" the number of customers, "Γ" the uncertainty budget, "Total (s)" the average total time, "Master (s)" the average time solving the master problem, "Adversarial (s)" the average time solving

			Total (s)			Master (s)			Adversarial (s)			Iterations			Count		
				CCG+...			CCG+...			CCG+...			CCG+...			CCG+...	
$\|V_1\|$	$\|V_2\|$	Γ	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H
10	20	3	**19.9**	417.7	513.5	**15.1**	412.4	508.2	2.8	2.6	**2.6**	**9.4**	**9.4**	**9.4**	**10**	**10**	**10**
10	20	4	**7.3**	178.5	172.1	**3.9**	174.7	168.5	**2.5**	2.6	2.5	**7.6**	**7.6**	**7.6**	**10**	**10**	**10**
10	30	3	**44.8**	464.6	416.3	**32.2**	451.5	402.5	**9.6**	9.7	10.3	**8.2**	**8.2**	**8.2**	**10**	**10**	**10**
10	30	4	**40.8**	634.2	558.4	**23.7**	616.6	540.2	12.6	**11.9**	12.9	12.8	**12.7**	**12.7**	**10**	**10**	**10**
10	40	3	**67.7**	664.4	481.2	**45.3**	626.3	444.5	**18.3**	28.3	27.4	9.4	**9.1**	**9.1**	8	**10**	**10**
10	40	4	**23.9**	851.3	744.6	**15.4**	803.9	693.2	**6.5**	31.4	33.9	**9.1**	10.5	10.5	8	**10**	**10**
15	20	3	**501.9**	1155.4	909.3	**484.6**	1136.9	891.4	14.8	14.2	**14.0**	**9.3**	9.6	9.5	9	9	**10**
15	20	4	**107.4**	2618.5	2497.9	**77.1**	2580.1	2461.0	**26.9**	30.1	29.6	**10.3**	11.4	11.4	7	**9**	**9**
15	30	3	**366.8**	2541.2	1905.4	**275.0**	2444.5	1741.7	**75.3**	80.4	147.8	10.7	**9.5**	**9.5**	**7**	6	6
15	30	4	**567.0**	4010.3	2862.5	**442.7**	3856.0	2702.8	**109.2**	133.4	139.1	12.4	**12.0**	12.4	**5**	**5**	**5**
15	40	3	**183.9**	5372.8	3714.3	**93.0**	5280.1	3551.7	85.7	**84.2**	152.6	9.2	9.2	**8.8**	5	5	**6**
15	40	4	**995.1**	4376.4	2583.5	**639.1**	4103.1	2315.3	307.1	235.0	**230.2**	**8.3**	8.5	9.0	3	**4**	**4**

TABLE 1. Solved Instances for the FLP. From left to right: the average total time, the average time spent solving the master problem, the average time spent solving the adversarial problem, the average number of iterations, and the number of solved instances.

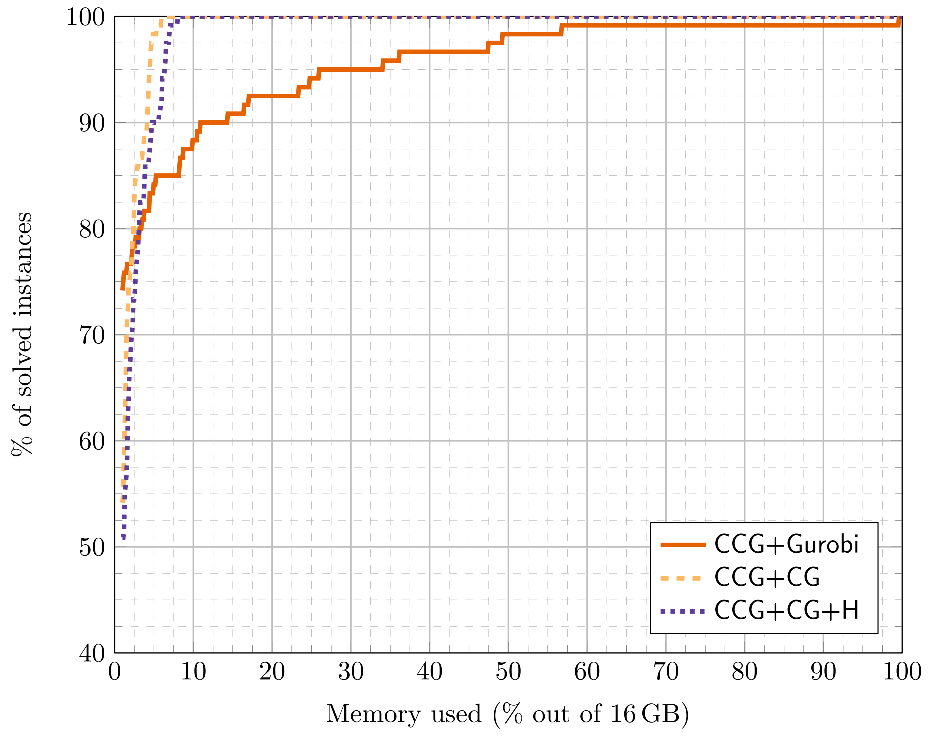

FIGURE 3. ECDFs of used memory for the FLP

			Iterations			Count		
			CCG+...			CCG+...		
$\lvert V_1\rvert$	$\lvert V_2\rvert$	Γ	Gurobi	CG	CG+H	Gurobi	CG	CG+H
10	40	3	5.5	—	—	2	—	—
10	40	4	9.0	—	—	2	—	—
15	20	3	11.0	5.0	—	1	1	—
15	20	4	**11.7**	15.0	15.0	3	1	**1**
15	30	3	**12.7**	13.5	14.3	**3**	4	4
15	30	4	13.8	**13.2**	13.4	**5**	**5**	**5**
15	40	3	**7.2**	7.6	10.0	5	5	**4**
15	40	4	**7.9**	8.8	11.2	7	**6**	**6**

TABLE 2. Unsolved Instances for the FLP. From left to right: the average number of iterations, and the number of unsolved instances.

the adversarial problem, "Iterations" the average number of generated scenarios instances, and "Count" the number of unsolved instances. Table 2 regards those instances which could not be solved within the time limit.

While the average total time for CCG+Gurobi seems impressively small compared to other approaches, it should be kept in mind that a larger number of instances is solved by CCG+CG and CCG+CG+H. In fact, as commented earlier in Section 3, the classic approach is limited to instances that require only a small number of scenarios to prove global optimality. For such problems, the computation time is very small. For example, for instances of size $(10, 40)$ and $\Gamma = 3$, the average computation time is 67.7 seconds for CCG+Gurobi and 664.4 seconds for CCG+CG.

However, CCG+Gurobi is unable to solve 2 such instances, with an average number of iterations of 5.5, while CCG+CG solves all instances with an average number of 9.1 iterations. This means that a lot more iterations are executed in the latter than in the first approach. This is also shown by a larger amount of time spent solving the adversarial problem in CCG+CG and CCG+CG+H compared to CCG+Gurobi. However, as usual, we see that a vast majority of the computation time is still spent on solving the master problem.

Finally, we can see that the use of the heuristic from Section 4.4 is beneficial for accelerating the computation time of the overall method since the average total computation time of CCG+CG+H represents 56 % of the one of CCG+CG for instances of size $(15, 40)$ with $\Gamma = 4$ while both methods can solve the same number of instances.

5.4. **Generalized Assignment Problem.**

5.4.1. *Problem Statement.* We consider a variant of the robust generalized assignment problem (GAP) introduced in Bodur et al. (2024). Let M be a set of machines and T be a set of tasks to be assigned to the machines. Each machine $m \in M$ has a maximum capacity b_m. For each task $t \in T$, we let w_t denote its weight and p_t its profit. The deterministic GAP can be modeled as

$$\begin{aligned}
\max_{y} \quad & \sum_{m \in M} \sum_{t \in T} p_t y_{mt} && \text{(18a)}\\
\text{s.t.} \quad & \sum_{t \in T} w_t y_{mt} \leq b_m, \quad m \in M, && \text{(18b)}\\
& \sum_{m \in M} y_{mt} \leq 1, \quad t \in T, && \text{(18c)}\\
& y_{mt} \in \{0,1\}, \quad m \in M, \, t \in T. && \text{(18d)}
\end{aligned}$$

Here, Constraints (18b) ensure that the capacity of each machine is not exceeded. Constraints (18c) enforce that a given task is assigned to at most one machine, while the objective function (18a) maximizes the profit for assigning the tasks to the machines.

We now consider an uncertain setting in which some assignments may become infeasible over time. Still, the decision maker aims to identify a set of potential machine-task pairings from which the actual assignments can be made once infeasible pairings become known. This situation can arise when there is a need to precondition the machines to perform a given task, if ever performed. Following Bodur et al. (2024), we assume to have a cardinality constraint limiting the number of pre-identified pairings. A model for the robust GAP reads

$$\begin{aligned}
\max_{x \in X} \; \min_{\xi \in \Xi} \; \max_{y} \quad & \sum_{m \in M} \sum_{t \in T} p_t y_{mt} && \text{(19a)}\\
\text{s.t.} \quad & \sum_{t \in T} w_t y_{mt} \leq b_m, \quad m \in M, && \text{(19b)}\\
& \sum_{m \in M} y_{mt} \leq 1, \quad t \in T, && \text{(19c)}\\
& y_{mt} \leq x_{mt}(1 - \xi_{mt}), \quad m \in M, \, t \in T, && \text{(19d)}\\
& y_{mt} \in \{0,1\}, \quad m \in M, \, t \in T, && \text{(19e)}
\end{aligned}$$

with

$$X := \left\{ x \in \{0,1\}^{|M| \times |T|} : \sum_{m \in M} \sum_{t \in T} x_{mt} \leq \beta \right\},$$

for a given budget β. Constraint (19d) ensures that only pre-identified pairings that remain feasible in a given scenario $\xi \in \Xi$ can be chosen. Here, Ξ is a set of scenarios chosen so that $\xi_{mt} = 1$ if and only if the pairing (m, t) is infeasible in scenario ξ. More precisely, we consider

$$\Xi := \left\{ \xi \in \{0,1\}^{|M|\times|T|} \colon \sum_{m\in M} \sum_{t\in T} \xi_{mt} \leq \Gamma \right\},$$

where Γ controls the maximum number of assignments that can become infeasible in a given scenario.

5.4.2. *Instances.* The instances are randomly generated in the same way as for the facility location application where facilities are interpreted as machines and tasks as clients. We consider instances with $|M| \in \{10, 15\}$ and $|T| \in \{20, 30, 40\}$. Additionally, we set $\beta = 0.6 \times |M| \times |T|$ and $\Gamma \in \{4, 5, 10\}$.

5.4.3. *Discussion of the Results.* For a time limit of 3 hours, Figure 4 depicts the ECDF of computation times over our test set for all three methods. It can be seen that both CCG+CG and CCG+CG+H outperform the classic CCG+Gurobi approach. Indeed, while the latter solves around 45 % of the test set, the CCG+CG+H approach solves around 57 % of the instances within the time limit. The impact of the CG-based heuristic is also demonstrated since roughly 47% of the instances are solved by CCG+CG. Similarly to the previous section, Figure 5 depicts the ECDF of memory used by each approach out of the 16 GB available. Again, it can be seen that using column generation to solve the master problem is less demanding in terms of memory usage than when using Gurobi. This seems to indicate that smaller search trees are needed when CG is used at every iteration.

Table 4 and Table 3 are analogous to Table 2 and Table 1 for the FLP application. Here, $|M|$ is the number of machines and $|T|$ the number of tasks. In column "Count" of Table 4, we reported between parentheses the number of instances which hit the memory limit. Note that running out of memory occurs when a too large search tree is needed to solve that instance. Hence, the chances of solving that instance with a larger amount of memory is also small. Clearly, using CCG+CG+H yields the best performance. This is exemplified by those instances with 10 machines, 30 tasks, and an uncertainty budget $\Gamma = 10$. For those instances, CC+Gurobi can solve only 4 instances out of 10 while CCG+CG+H solves 7 instances within the time limit. Moreover, the average time spent solving the master problem is more than 13 times worse for Gurobi than with column generation and its heuristic.

5.5. **Job Scheduling with Uncertain Processing Time.**

5.5.1. *Problem Statement.* We now consider a scheduling application based on the deterministic problem denoted $1|r_j|\sum w_j U_j$ in the literature (Pinedo 2008). This problem considers a single machine on which a set of jobs with release and due dates have to be scheduled while minimizing the weighted number of tardy jobs. To formally state this problem, let us consider a finite set of jobs J. For each job $j \in J$, let p_j denote its processing time, r_j its release date, d_j its due date, and w_j its weight, which is the penalty to be paid if the job is performed tardily. Following the approach in Detienne (2014), we model the deterministic version of the problem as a MIP. This MIP formulation is based on that jobs with agreeable time windows can be scheduled according to the earliest-deadline-first rule, i.e., if for two jobs $i, j \in J$ it holds $r_i \leq r_j$ and $d_i \leq d_j$, then there is an optimal schedule for $1|r_j|\sum w_j U_j$ such that i is scheduled before j. The key idea is then to reformulate the scheduling problem into the problem of selecting jobs with agreeable time windows. To this end, for each pair of job $(i, j) \in J^2$ such that i and j do not have agreeable time

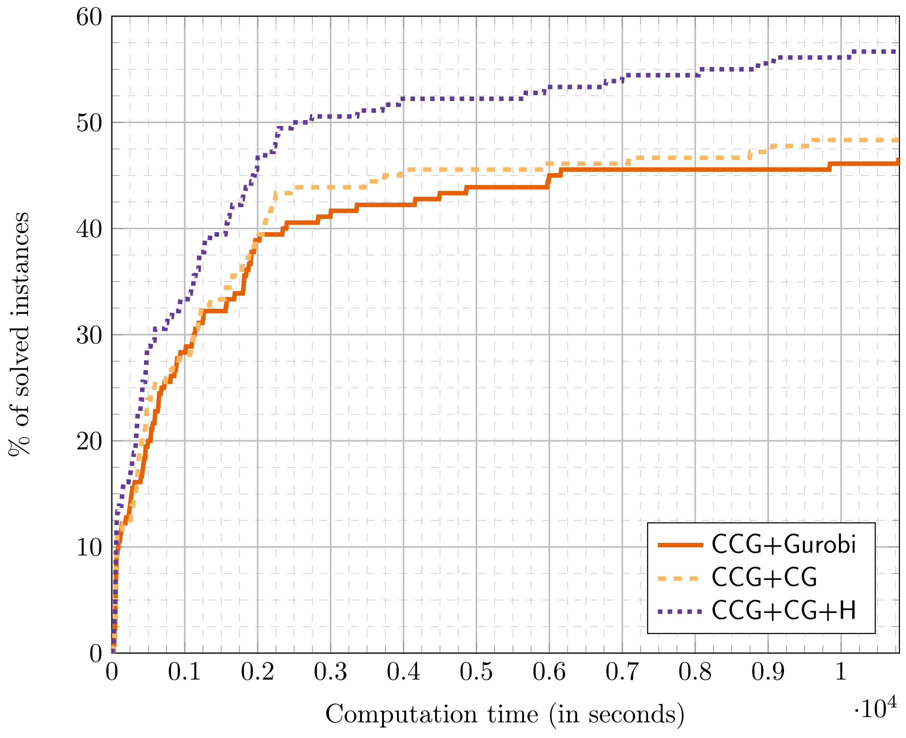

FIGURE 4. ECDFs of computation times for the GAP

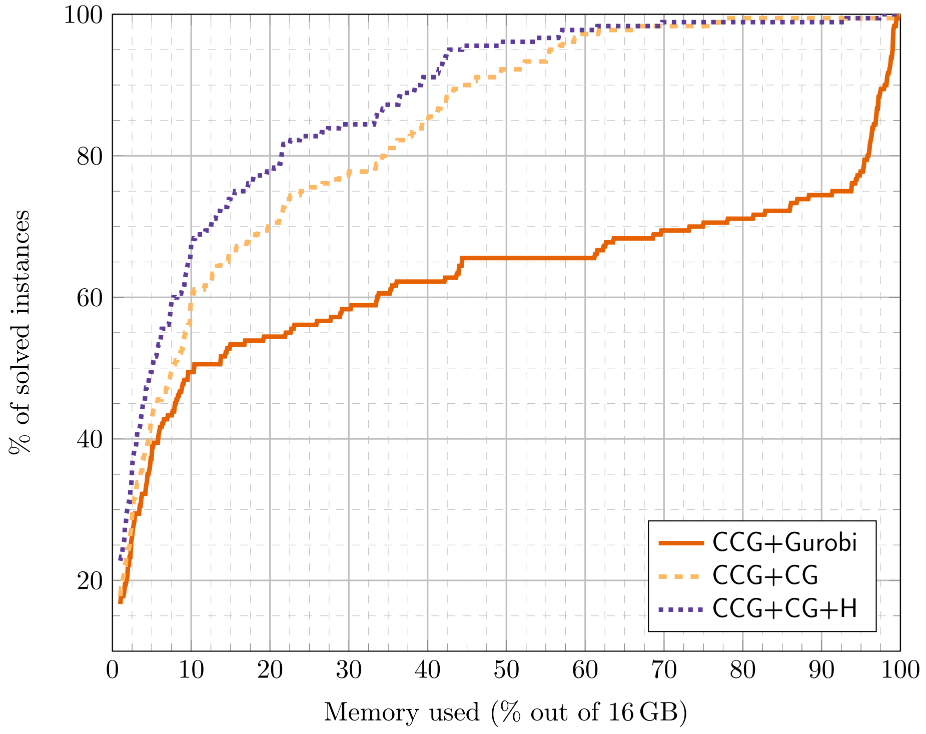

FIGURE 5. ECDFs of used memory for the GAP

			Total (s)			Master (s)			Adversarial (s)			Iterations			Count		
			CCG+...			CCG+...			CCG+...			CCG+...			CCG+...		
$\|M\|$	$\|T\|$	Γ	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H
10	20	4	**133.9**	135.6	135.2	2.7	**1.0**	**1.0**	126.9	110.2	**110.0**	25.4	**11.2**	**11.2**	**10**	**10**	**10**
10	20	5	—	3479.0	**2479.2**	—	764.5	**9.0**	—	2672.1	**2451.2**	—	66.1	**50.4**	—	**8**	7
10	20	10	596.9	651.4	**251.9**	40.2	511.6	**1.8**	537.2	**138.1**	247.7	52.3	**15.0**	19.3	9	4	**10**
10	30	4	1477.0	**337.4**	344.7	459.0	0.1	**0.1**	910.4	**337.3**	344.6	102.4	**2.0**	**2.0**	9	**10**	**10**
10	30	5	—	8068.0	**7927.3**	—	0.1	**0.1**	—	8067.8	**7927.1**	—	**3.0**	**3.0**	—	**2**	**2**
10	30	10	808.5	**299.4**	1830.5	33.0	26.7	**2.5**	748.0	**271.5**	1823.1	47.5	**11.0**	22.0	4	1	**7**
10	40	4	4612.7	**1647.5**	1672.8	2383.1	0.1	**0.1**	1858.1	**1647.4**	1672.7	188.0	**2.0**	**2.0**	3	**10**	**10**
10	40	10	**5538.2**	—	8285.5	22.3	—	**3.3**	**5493.5**	—	8275.6	44.0	—	**24.0**	**4**	—	3
15	20	4	53.1	**45.5**	45.7	0.4	0.1	**0.1**	51.5	**45.4**	45.6	12.1	**2.0**	**2.0**	**10**	**10**	**10**
15	20	5	**891.6**	992.8	995.0	1.0	0.1	**0.1**	**887.5**	992.7	994.9	19.3	**2.0**	**2.0**	**10**	**10**	**10**
15	20	10	367.5	315.9	**188.5**	0.7	292.7	**1.6**	364.5	**22.8**	184.0	17.0	**5.0**	15.5	**2**	1	**2**
15	30	4	459.3	442.9	**441.5**	1.3	0.1	**0.1**	454.5	442.8	**441.4**	17.1	**2.0**	**2.0**	**10**	**10**	**10**
15	30	5	**2885.1**	5925.8	5927.3	5.8	0.1	**0.1**	**2868.3**	5925.7	5927.1	29.7	**2.0**	**2.0**	**3**	1	1
15	40	4	3209.1	2527.6	**2522.8**	2.0	0.1	**0.1**	3201.9	2526.1	**2521.3**	17.8	**2.3**	**2.3**	**10**	**10**	**10**

TABLE 3. Solved Instances for the GAP. From left to right: the average total time, the average time spent solving the master problem, the average time spent solving the adversarial problem, the average number of iterations, and the number of solved instances.

			Iterations			Count		
			CCG+...			CCG+...		
$\lvert M\rvert$	$\lvert T\rvert$	Γ	Gurobi	CG	CG+H	Gurobi	CG	CG+H
10	20	5	319.6	**187.5**	189.0	10 (9)	**2** (1)	3 (2)
10	20	10	320.0	**20.0**	—	1 (1)	6	—
10	30	4	261.0	—	—	1 (1)	—	—
10	30	5	256.3	**43.1**	**43.1**	10 (8)	**8**	**8**
10	30	10	183.8	20.0	**12.3**	6 (3)	9	**3**
10	40	4	225.0	—	—	7 (7)	—	—
10	40	5	225.3	31.3	**30.8**	**10** (10)	**10**	**10**
10	40	10	137.8	**22.9**	23.1	**6** (1)	10	7
15	20	10	190.1	**70.2**	75.6	**8** (2)	9	**8**
15	30	5	30.7	**2.0**	**2.0**	**7**	9	9
15	30	10	108.1	52.1	**51.7**	**10** (1)	**10**	**10**
15	40	5	57.3	**1.9**	**1.9**	**10** (2)	**10** (1)	**10** (1)
15	40	10	35.8	**37.7**	37.8	**10**	**10**	**10**

Table 4. Unsolved Instances for the GAP. From left to right: the average number of iterations, and the number of unsolved instances. Between parenthesis, the number of instances which went out of memory.

windows, a new job occurrence k is created, with $d_k = d_j, r_k = r_i$, $p_k = p_i$, and $w_k = 0$, which yields that i is scheduled before j. The original job is also added to the set of job occurrences with a weight of 0. We let $\tilde{J}$ be the set of all job occurrences and G_j be the set of all job occurrences related to a given job $j \in J$. Minimizing the weighted number of tardy jobs can then be seen as selecting, for each job $j \in J$, a job occurrence $k \in G_j$ or paying a penalty cost of w_j. We refer the interested reader to Detienne (2014) for more details. The corresponding MIP model reads

$$\min_{x,y,t} \quad \sum_{j\in J} w_j x_j \tag{20a}$$

$$\text{s.t.} \quad \sum_{k\in G_j} y_k = 1 - x_j, \quad j \in J, \tag{20b}$$

$$t_k \le d_k, \quad k \in \tilde{J}, \tag{20c}$$

$$t_k - t_{k-1} - p_j y_k \ge 0, \quad k \in G_j,\ j \in J, \tag{20d}$$

$$t_k - p_j y_k - M_k y_k \ge r_k - M_k, \quad k \in G_j,\ j \in J, \tag{20e}$$

$$y_k \in \{0,1\}, t_k \ge 0, \quad k \in G_j,\ j \in J, \tag{20f}$$

$$x_j \in \{0,1\}, \quad j \in J. \tag{20g}$$

Here, variables x are used to model tardy executions of tasks, i.e., for each job $j \in J$, $x_j = 1$ holds if and only if j is executed tardily. Variables y are used to represent the selection of a job occurrence while t is used for computing the termination time of a job. Constraints (20b) enforce that a job occurrence must be selected if and only if the job is scheduled on time. Constraints (20c) force scheduled jobs to terminate before their deadlines while Constraints (20d) enforce release dates. Finally, Constraints (20e) make sure that no overlapping of jobs occurs in a feasible schedule.

We now introduce uncertainty to this scheduling problem. To this end, the set of jobs to be performed has to be decided here and now, while the exact processing time of the jobs is not perfectly known. Formally, we assume that each job $j \in J$ may have a processing time p_j being randomly increased by δ_j. More formally, we rely on the Γ-uncertainty set $\Xi := \{\xi \in \{0,1\}^{|J|} \colon \sum_{j\in J} \xi_j \leq \Gamma\}$, where for any $\xi \in \Xi$, $\xi_j = 1$ means that job $j \in J$ has a processing time of $p_j + \delta_j$ and p_j otherwise. The goal is to select a subset of jobs to be performed so as to minimize the penalties. Performing a job $j \in J$ on time is associated to a reward of f_j, while performing it tardily implies a penalty of w_j.

The ARO version of the problem reads

$$\min_{x\in\{0,1\}^{|J|}} \left\{ \sum_{j\in J} w_j x_j + \max_{\xi\in\Xi} \min_{y\in Y(x,\xi)} \sum_{j\in J} -(w_j + f_j) \sum_{k\in G_j} y_k \right\},$$

in which $Y(x,\xi)$ is chosen so that $y \in Y(x,\xi)$ holds if and only if there exists $t \in \mathbb{R}^{|\tilde{J}|}$ satisfying

$$\begin{aligned}
&\sum_{k\in G_j} y_k \leq x_j, \quad j \in J,\\
&t_k \leq d_k, \quad k \in \tilde{J},\\
&t_k - t_{k-1} - (p_j + \delta_j\xi_j)y_k \geq 0, \quad k \in G_j,\ j \in J,\\
&t_k - (p_j + \delta_j\xi_j)y_k - M_k y_k \geq r_k - M_k, \quad k \in G_j,\ j \in J,\\
&y_k \in \{0,1\},\ t_k \geq 0, \quad k \in G_j,\ j \in J.
\end{aligned}$$

This problem can be seen as a variant of the adjustable robust problem studied in Clautiaux et al. (2023). However, the problem we consider is different both w.r.t. its mathematical structure and its interpretation. In Clautiaux et al. (2023), it is assumed that jobs may fail in such a way that the reward of a job is decreased. Restoring the "full" nominal reward is possible by increasing the processing time of the job. This is interpreted as "repairing" the failed task and must be decided. Mathematically, it is an ARO problem with uncertain cost and a continuous uncertainty set. In our problem, the processing time of a job is directly affected by the uncertainty, which is modeled using a discrete set. Thus, a major difference from the computational viewpoint is that our problem has uncertain constraints. Finally, note that our problem is easily shown to be Σ_2^P-hard by reduction to the knapsack interdiction problem (Caprara et al. 2016), while Clautiaux et al. (2023) showed NP-completeness of their problem.

5.5.2. *Instances.* All instances are randomly generated according to Dauzère-Pérès and Sevaux (2002) using the parameters N, R, and D defined as the number of jobs, the control for dispersion of release dates, and the control for dispersion of deadlines, respectively. For each job $j \in J$, the processing time p_j and the weight w_j are uniformly generated between 0 and 100. The maximum increase in processing time δ_j is set to $20\,\% \times p_j$. The release date r_j is drawn from $[0, NR]$ while the deadline d_j is set to $r_j + p_j + \Delta_j$, where Δ_j is a random number uniformly drawn from $[0, ND]$.

Our test set consists of instances with 30 and 40 jobs with R and D taking the values $\{1, 5, 10, 20\}$. For each combination of these parameters, we generate 10 instances, yielding a total of 160 nominal instances. Each instance is considered with an uncertainty budget $\Gamma \in \{3, 6\}$ leading to 320 robust instances.

5.5.3. *Discussion of the Results.* For a time limit of 3 hours, Figure 6 depicts the ECDF of computation times over the set of instances for all three methods. Contrary

		Total (s)			Master (s)			Adversarial (s)			Iterations			Count		
			CCG+…			CCG+…			CCG+…			CCG+…			CCG+…	
$\|J\|$	Γ	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H	Gurobi	CG	CG+H
30	3	**158.5**	2010.1	1821.5	**130.2**	1986.2	1798.7	28.2	23.0	**22.2**	8.5	7.2	**7.1**	**71**	62	61
30	6	**560.3**	1481.4	1394.1	**80.4**	1239.0	1129.5	479.9	**241.8**	264.1	6.7	**6.6**	6.7	**59**	53	52
40	3	**315.3**	1469.2	1533.4	**207.9**	1438.3	1505.4	107.2	29.6	**27.0**	10.5	**7.8**	**7.8**	**48**	19	19
40	6	**299.5**	1764.4	1471.4	**197.6**	1746.9	1454.2	101.8	**15.0**	15.7	9.3	**8.3**	**8.3**	**30**	24	24

TABLE 5. Solved Instances for the JSP. From left to right: the average total time, the average time spent solving the master problem, the average time spent solving the adversarial problem, the average number of iterations, and the number of solved instances.

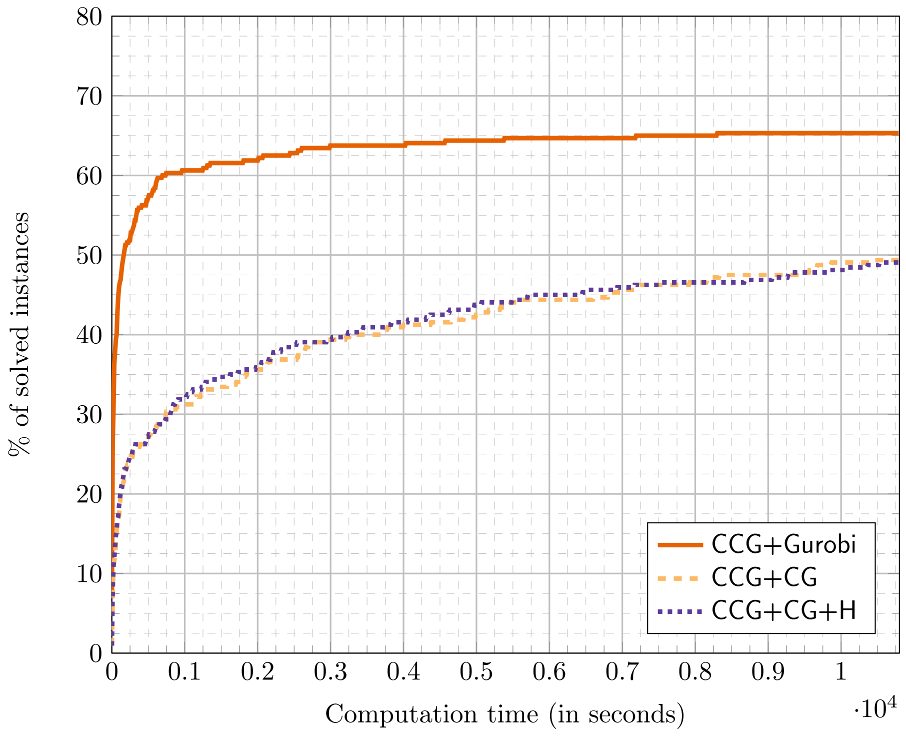

FIGURE 6. ECDFs of computation times for the JSP

		Iterations			Count		
		CCG+...			CCG+...		
$\|J\|$	Γ	Gurobi	CG	CG+H	Gurobi	CG	CG+H
30	3	**6.9**	6.1	6.5	**9**	18	19
30	6	**8.6**	6.1	6.0	**21**	27	28
40	3	**8.9**	5.7	5.8	**32**	61	61
40	6	**6.5**	3.8	3.9	**50**	56.00	56.00

TABLE 6. Unsolved Instances for the JSP. From left to right: the average number of iterations, and the number of unsolved instances.

to the previous two applications, it can be seen that the classic CCG+Gurobi approach outperforms the CG-enhanced approaches. Indeed, while Gurobi is able to solve 65 % of the instance set, CCG+CG only solves 50 % of the instances within the time limit. This can be explained by at least two factors. First, the pricing problem in the scheduling application is harder to solve than those in the FLP or GAP applications. For instance, both the pricing problem for the FLP and that of the GAP are purely binary problems while that of the JSP is a mixed-integer problem. Moreover, the number of variables and constraints are linear in the size of the instance for the FLP and the GAP while it is quadratic for the JSP application; recall that $|\tilde{J}| \in \mathcal{O}(|J|^2)$. Hence, even though the dual bound associated to the CG-based approaches are stronger, solving each node within the branch-and-price algorithm becomes very challenging. Second, the master problem solved at each step of CCG+CG suffers from a high dual degeneracy, resulting in a lot of columns which needs to be added to make progress in the CG algorithm. Indeed, for a fixed scenario, many columns can be seen as equivalent since columns are associated to specific orderings of the jobs while costs only reflects jobs which are effectively scheduled on time, regardless of their ordering. More details are given in Table 5 and Table 6 which are similar to the previous tables presented for the FLP and the GAP application.

6. Conclusion

In this paper, we introduced a new and stronger formulation of the master problem used in the classic CCG method, which is a standard approach in ARO. We showed how this model can be solved by exploiting CG techniques to obtain an efficient method for a specific but large class of problems. We numerically evaluated the novel method applied to three different applications. While for two out of them, the novel CG-based approaches outperform the vanilla CCG method, this is not the case for the scheduling application. However, our discussion of the potential reasons for this may give the user a hint to decide when or when not to use our novel approaches.

Let us conclude with some remarks regarding the limitations of the novel approach and some potential future research directions. A clear limitation of the method presented in this paper is Assumption 5. Indeed, it is necessary to prove the main theorem and is therefore essential for the exactness of the overall algorithm. We would like to point out that Assumption 5 can theoretically be relaxed by a weaker assumption, which just claims that there exists an index set $I \subseteq \{1, \dots, n_x\}$ such that $x \in X$ implies $x_I \in \{0, 1\}^{|I|}$ and that $Y(x, \xi) = Y(x_I, \xi)$ holds. Under the latter assumption, one can still apply our derivations by working on a reformulated problem, which is given by

$$\min_{x \in X} \left\{ c^\top x + \max_{\xi \in \Xi} \min_{y \in Y^+(x,\xi)} d(\xi)^\top y \right\}$$

with

$$Y^+(x, \xi) = \left\{ y \colon \exists x' \in [0, 1]^{|I|} \text{ with } y \in Y(x', \xi),\ x' \le x,\ x' \ge x \right\}.$$

Indeed, we then have

$$\{x_I\} \times \operatorname{conv}(Y^+(x, \xi)) = \operatorname{conv}(Y([0, 1]^{|I|}, \xi)) \cap \{(x', y) \colon x' = x_I\}$$

for any $x \in X$. This has been suggested by Arslan and Detienne (2022), who work with a similar assumption in the context of cost-uncertain ARO. It has also been used by Sherali and Fraticelli (2002) in the context of two-stage stochastic optimization. Unfortunately, some preliminary computational experiments showed poor performance of this approach, which can be explained by the increased difficulty of the pricing problem within CG. Indeed, the pricing problem is now expressed in a higher-dimensional space, which considerably slows down its solution time and, thus, the total computation time. As a result, the benefit of using CG to solve the master problem vanishes.

Consequently, a possible future research direction is to study efficient methods to handle constraints such as "$y \in \operatorname{conv}(Y(x, \xi))$" using weaker assumptions. Moreover, let us point out that our method is designed for problems with mixed-integer second-stage decisions. While our method can be applied as-is to problems with purely continuous second-stage decisions, it is unlikely to perform better than the usual CCG in this case. This is because convex problems do not suffer from the computational challenges discussed in Section 3, which are due to the quality of the continuous relaxation of the underlying ARO problem, which is much stronger if no integer variables appear in the second stage.

Another important aspect for future work is to develop solvers for bilevel problems of the type in (1); see Assumption 4. While this is out of the scope of this paper, any progress in this direction will also improve the applicability of the CG-based approaches discussed in this paper.

Finally, there are many other highly relevant problem classes such as k-resilience problems (Pfetsch and Schmitt 2023) or fortification games (Leitner et al. 2023; Lozano and Smith 2017) to which our method might be extended to since they have a structure similar to the one studied in this paper.

Acknowledgements

We acknowledge the support by the German Bundesministerium für Bildung und Forschung within the project "RODES" (Förderkennzeichen 05M22UTB). The computations were executed on the high performance cluster "Elwetritsch" at the TU Kaiserslautern, which is part of the "Alliance of High Performance Computing Rheinland-Pfalz" (AHRP). We kindly acknowledge the support of RHRK.

References

Achterberg, T. (2009). "SCIP: solving constraint integer programs." In: *Mathematical Programming Computation* 1.1, pp. 1–41. DOI: 10.1007/s12532-008-0001-1.

Agra, A., M. Christiansen, R. Figueiredo, L. M. Hvattum, M. Poss, and C. Requejo (2013). "The robust vehicle routing problem with time windows." In: *Computers & Operations Research* 40.3. Transport Scheduling, pp. 856–866. DOI: 10.1016/j.cor.2012.10.002.

Arslan, A. N. and B. Detienne (2022). "Decomposition-Based Approaches for a Class of Two-Stage Robust Binary Optimization Problems." In: *INFORMS Journal on Computing* 34.2, pp. 857–871. DOI: 10.1287/ijoc.2021.1061.

Ben-Tal, A., A. Goryashko, E. Guslitzer, and A. Nemirovski (2004). "Adjustable robust solutions of uncertain linear programs." In: *Mathematical Programming* 99.2, pp. 351–376. DOI: 10.1007/s10107-003-0454-y.

Ben-Tal, A., S. Bhadra, C. Bhattacharyya, and J. Saketha Nath (2011). "Chance constrained uncertain classification via robust optimization." In: *Mathematical Programming* 127.1, pp. 145–173. DOI: 10.1007/s10107-010-0415-1.

Ben-Tal, A., L. E. Ghaoui, and A. Nemirovski (2009). *Robust Optimization.* Princeton University Press. DOI: 10.1515/9781400831050.

Bertsimas, D. and C. Caramanis (2010). "Finite Adaptability in Multistage Linear Optimization." In: *IEEE Transactions on Automatic Control* 55.12, pp. 2751–2766. DOI: 10.1109/tac.2010.2049764.

Bertsimas, D. and V. Goyal (2011). "On the power and limitations of affine policies in two-stage adaptive optimization." In: *Mathematical Programming* 134.2, pp. 491–531. DOI: 10.1007/s10107-011-0444-4.

Bertsimas, D., D. A. Iancu, and P. A. Parrilo (2010). "Optimality of Affine Policies in Multistage Robust Optimization." In: *Mathematics of Operations Research* 35.2, pp. 363–394. DOI: 10.1287/moor.1100.0444.

Bertsimas, D. and S. Shtern (2018). *A Scalable Algorithm for Two-Stage Adaptive Linear Optimization.* DOI: 10.48550/ARXIV.1807.02812.

Bertsimas, D. and M. Sim (2004). "The Price of Robustness." In: *Operations Research* 52.1, pp. 35–53. DOI: 10.1287/opre.1030.0065.

Bodur, M., T. C. Y. Chan, and I. Y. Zhu (2024). *Network Flow Models for Robust Binary Optimization with Selective Adaptability.* DOI: 10.48550/ARXIV.2403.19471.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization.* Cambridge University Press. DOI: 10.1017/CBO9780511804441.

Caprara, A., M. Carvalho, A. Lodi, and G. J. Woeginger (2016). “Bilevel Knapsack with Interdiction Constraints.” In: *INFORMS Journal on Computing* 28.2, pp. 319–333. DOI: 10.1287/ijoc.2015.0676.

Clautiaux, F., B. Detienne, and H. Lefebvre (2023). “A two-stage robust approach for minimizing the weighted number of tardy jobs with objective uncertainty.” In: *Journal of Scheduling* 26.2, pp. 169–191. DOI: 10.1007/s10951-022-00775-1.

Dauzère-Pérès, S. and M. Sevaux (Dec. 2002). “Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine.” In: *Naval Research Logistics (NRL)* 50.3, pp. 273–288. DOI: 10.1002/nav.10056.

Desrosiers, J. and M. E. Lübbecke (2005). “A Primer in Column Generation.” In: *Column Generation*. Springer-Verlag, pp. 1–32. DOI: 10.1007/0-387-25486-2_1.

Detienne, B. (2014). “A mixed integer linear programming approach to minimize the number of late jobs with and without machine availability constraints.” In: *European Journal of Operational Research* 235.3, pp. 540–552. DOI: 10.1016/j.ejor.2013.10.052.

Detienne, B., H. Lefebvre, E. Malaguti, and M. Monaci (2024). “Adjustable robust optimization with objective uncertainty.” In: *European Journal of Operational Research* 312.1, pp. 373–384. DOI: 10.1016/j.ejor.2023.06.042.

Fischetti, M., I. Ljubić, M. Monaci, and M. Sinnl (2019). “Interdiction Games and Monotonicity, with Application to Knapsack Problems.” In: *INFORMS Journal on Computing* 31.2, pp. 390–410. DOI: 10.1287/ijoc.2018.0831.

Gabrel, V., C. Murat, and A. Thiele (2014). “Recent advances in robust optimization: An overview.” In: *European Journal of Operational Research* 235.3, pp. 471–483. DOI: 10.1016/j.ejor.2013.09.036.

García, R., A. Marín, and M. Patriksson (2003). “Column Generation Algorithms for Nonlinear Optimization, I: Convergence Analysis.” In: *Optimization* 52.2, pp. 171–200. DOI: 10.1080/0233193031000079856.

Gurobi Optimization, LLC (2023). *Gurobi Optimizer Reference Manual*. URL: https://www.gurobi.com.

Hanasusanto, G. A., D. Kuhn, and W. Wiesemann (2015). “K-Adaptability in Two-Stage Robust Binary Programming.” In: *Operations Research* 63.4, pp. 877–891. DOI: 10.1287/opre.2015.1392.

Kämmerling, N. and J. Kurtz (2020). “Oracle-based algorithms for binary two-stage robust optimization.” In: *Computational Optimization and Applications* 77.2, pp. 539–569. DOI: 10.1007/s10589-020-00207-w.

Kurtz, J. (2023). *Approximation Algorithms for Min-max-min Robust Optimization and K-Adaptability under Objective Uncertainty*. arXiv: 2106.03107 [math.OC].

Lefebvre, H. (2023). *idol: Generic decomposition methods for mathematical programming*. publicly available online. URL: https://hlefebvr.github.io/idol/ (visited on 09/18/2023).

Lefebvre, H., E. Malaguti, and M. Monaci (2023). “Adjustable Robust Optimization with Discrete Uncertainty.” In: *INFORMS Journal on Computing*. DOI: 10.1287/ijoc.2022.0086.

Lefebvre, H. and A. Subramanyam (2024). *Correction to: A Lagrangian dual method for two-stage robust optimization with binary uncertainties*. arXiv: 2411.04307 [math.OC].

Leitner, M., I. Ljubić, M. Monaci, M. Sinnl, and K. Tanınmış (2023). “An exact method for binary fortification games.” In: *European Journal of Operational Research* 307.3, pp. 1026–1039. DOI: 10.1016/j.ejor.2022.10.038.

Lozano, L. and J. C. Smith (2017). “A Backward Sampling Framework for Interdiction Problems with Fortification.” In: *INFORMS Journal on Computing* 29.1, pp. 123–139. DOI: 10.1287/ijoc.2016.0721.

Meyer, R. R. (1974). "On the existence of optimal solutions to integer and mixed-integer programming problems." In: *Mathematical Programming* 7.1, pp. 223–235. DOI: 10.1007/bf01585518.

Pessoa, A., R. Sadykov, E. Uchoa, and F. Vanderbeck (2013). "In-Out Separation and Column Generation Stabilization by Dual Price Smoothing." In: *Experimental Algorithms*. Springer Berlin Heidelberg, pp. 354–365. DOI: 10.1007/978-3-642-38527-8_31.

Pfetsch, M. E. and A. Schmitt (2023). "A generic optimization framework for resilient systems." In: *Optimization Methods and Software* 38.2, pp. 356–385. DOI: 10.1080/10556788.2022.2142581.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall international series in industrial and systems engineering. Springer. DOI: 10.1007/978-1-4614-2361-4.

Rockafellar, R. T. (1970). *Convex Analysis*. Princeton University Press. DOI: 10.1515/9781400873173.

Sherali, H. D. and B. M. Fraticelli (2002). "A modification of Benders' decomposition algorithm for discrete subproblems: An approach for stochastic programs with integer recourse." In: *Journal of Global Optimization* 22.1, pp. 319–342. DOI: 10.1023/A:1013827731218.

Subramanyam, A. (2022). "A Lagrangian dual method for two-stage robust optimization with binary uncertainties." In: *Optimization and Engineering* 23.4, pp. 1831–1871. DOI: 10.1007/s11081-022-09710-x.

Subramanyam, A., C. E. Gounaris, and W. Wiesemann (2019). "K-adaptability in two-stage mixed-integer robust optimization." In: *Mathematical Programming Computation* 12.2, pp. 193–224. DOI: 10.1007/s12532-019-00174-2.

Sun, X. A. and A. J. Conejo (2021). *Robust Optimization in Electric Energy Systems*. Springer International Publishing. DOI: 10.1007/978-3-030-85128-6.

Tahernejad, S., T. K. Ralphs, and S. T. DeNegre (2020). "A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation." In: *Mathematical Programming Computation* 12.4, pp. 529–568. DOI: 10.1007/s12532-020-00183-6.

Tardella, F. (2004). "On the existence of polyhedral convex envelopes." In: *Nonconvex Optimization and Its Applications*. Springer US, pp. 563–573. DOI: 10.1007/978-1-4613-0251-3_30.

Wolsey, L. (1998). *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley. DOI: 10.1002/9781119606475.

Xidonas, P., R. Steuer, and C. Hassapis (2020). "Robust portfolio optimization: a categorized bibliographic review." In: *Annals of Operations Research* 292.1, pp. 533–552. DOI: 10.1007/s10479-020-03630-8.

Zeng, B. and L. Zhao (2013). "Solving two-stage robust optimization problems using a column-and-constraint generation method." In: *Operations Research Letters* 41.5, pp. 457–461. DOI: 10.1016/j.orl.2013.05.003.

Zhao, L. and B. Zeng (2012). *An Exact Algorithm for Two-stage Robust Optimization with Mixed Integer Recourse Problems*. URL: https://optimization-online.org/2012/01/3310/.

Zhen, J., D. den Hertog, and M. Sim (2018). "Adjustable Robust Optimization via Fourier–Motzkin Elimination." In: *Operations Research* 66.4, pp. 1086–1100. DOI: 10.1287/opre.2017.1714.

(H. Lefebvre, M. Schmidt, J. Thürauf) Trier University, Department of Mathematics, Universitätsring 15, 54296 Trier, Germany

Email address: lefebvre@uni-trier.de

Email address: martin.schmidt@uni-trier.de

(J. Thürauf) University of Technology Nuremberg (UTN), Department Liberal Arts and Social Sciences, Discrete Optimization Lab, Dr.-Luise-Herzberg-Str. 4, 90461 Nuremberg, Germany
Email address: `johannes.thuerauf@utn.de`