A SOLUTION ALGORITHM FOR CHANCE-CONSTRAINED PROBLEMS WITH INTEGER SECOND-STAGE RECOURSE DECISIONS*

ANDREA LODI†, ENRICO MALAGUTI‡, MICHELE MONACI‡, GIACOMO NANNICINI§, AND PAOLO PARONUZZI‡

Abstract. We study a class of chance-constrained two-stage stochastic optimization problems where the second-stage recourse decisions belong to mixed-integer convex sets. Due to the nonconvexity of the second-stage feasible sets, standard decomposition approaches cannot be applied. We develop a provably convergent branch-and-cut scheme that iteratively generates valid inequalities for the convex hull of the second-stage feasible sets, resorting to spatial branching when cutting no longer suffices. We show that this algorithm attains an approximate notion of convergence, whereby the feasible sets are relaxed by some positive tolerance ϵ. Computational results on chance-constrained resource planning problems indicate that our implementation of the proposed algorithm is highly effective in solving this class of problems, compared to a state-of-the-art MIP solver and to a naive decomposition scheme.

Key words. Chance-constrained mathematical program; Outer approximation; Branch-and-Cut; Convergence analysis; Computational experiments

AMS subject classifications. 68Q25, 68R10, 68U05

*Submitted to the editors DATE.

Funding: Andrea Lodi was funded by the Fog Research Institute under contract no. FRI-454. Enrico Malaguti, Michele Monaci, and Paolo Paronuzzi were funded by the Air Force Office of Scientific Research under awards number FA8655-20-1-7012 and FA8655-20-1-7019.

†CERC, Polytechnique Montréal, Canada

‡DEI "Guglielmo Marconi", Università di Bologna, Italy

§IBM Quantum, IBM T. J. Watson, USA

1. Introduction. We study the solution of *two-stage chance-constrained mathematical optimization problems*, formulated as:

$$\min\{cx : \Pr(x \in C_x(\omega)) \geq 1 - \alpha, x \in X\}, \tag{CCP}$$

where ω is a random variable with sample space Ω, $\alpha \in [0, 1]$, $X \subseteq \mathbb{R}^{n_x}$ is compact, and $C_x(\omega) = \text{Proj}_x C_{x,y}(\omega) \subseteq \mathbb{R}^{n_x}$ is the projection onto the x-space of a higher-dimensional set $C_{x,y}(\omega) \subseteq \mathbb{R}^{n_x+n_y(\omega)}$, whose dimension might depend on the realization ω. An algorithmic approach for optimizing over this problem class is a powerful tool for decision making under uncertainty: the problem class models situations where we want to make decisions that are feasible with high probability, as determined by $1 - \alpha$, and depending on the realization of uncertainty, we are allowed to take recourse actions (the y variables) to amend the initial decision x. Unlike two-stage stochastic programs [3], problem (CCP) does not require the first-stage solution to be feasible for all second-stage problems, but only for enough realizations to satisfy the chance constraint.

Chance-constrained problems are difficult to solve in general due to nonconvexity of the feasible region, and additional assumptions are often imposed in order to make the problem tractable. The importance of establishing conditions that guarantee existence of an equivalent deterministic problem has been identified since the early days of stochastic programming [5, 23]. In this paper we make a few assumptions that are common in the literature and guarantee the existence of a deterministic equivalent formulation; the most notable assumptions are that X is a bounded set and ω is a discrete random variable with finite support. The realizations of ω are called *scenarios*. For more general random variables, sample-average approximation provides a possible way to reduce to the case studied in this paper [20].

There are many ways to construct deterministic equivalent problems, under suitable conditions. A recent survey on reformulations of chance-constrained mixed-integer linear programs that arise from finite discrete distribution optimization can be found in [14]. The approach used in this paper leads to a mixed-integer program with one binary variable for each scenario, and big-M inequalities to describe the sets $C_x(\omega)$. The drawback of this approach is that the big-M inequalities could lead to poor bounds for the continuous relaxation. Strong valid inequalities for similar formulations under right-hand side uncertainty are discussed in [13, 21] and combine inequalities that are valid for single scenarios into so-called mixing inequalities. The idea is extended in [12], where aggregated mixing inequalities, incorporating lower bounds on the continuous variables in the original inequalities, are introduced. An alternative reformulation for the same class of problems can be constructed using the concept of $(1 - \epsilon)$-efficient points, see [24, 2, 6, 26]. Problem-specific algorithms can also be devised, e.g., the algorithm for chance-constrained packing problems developed in [28] using probabilistic covers.

For problems of the form (CCP) that admit a deterministic equivalent formulation including all scenarios, the direct solution of the deterministic equivalent quickly becomes intractable as the size of the problem and number of scenarios increase. Decomposition strategies that deal with each scenario separately have been shown to be very effective in this context and in multi-stage programming general, but their applicability depends on the structure of the problem at hand. If the second-stage feasible sets $C_{x,y}(\omega)$ are polyhedra with an explicitly known description (i.e., linear programs), we can apply Benders decomposition [1] or the L-shaped method [29]; Benders decomposition is at the heart of effective branch-and-cut algorithms for chance-constrained two-stage problems [19, 17]. However, this approach cannot be applied, in general, if the second-stage sets $C_{x,y}(\omega)$ are not polyhedra, because finding an appropriate outer approximation of these sets is no longer as simple as applying linear programming duality. The convexification of second-stage

problems containing binary variables only is studied in [27], using disjunctive programming and the facial property of 0-1 programs. For the linear case in which the first-stage problem contains binary variables only and the second stage includes integer variables, [8] proposes a decomposition algorithm that relies on parametric Gomory cuts, while [30] extends the approach to deal with first-stage general integer variables. The case with mixed-integer first and second-stage variables is considered in [25], where the solution algorithm combines branch-and-bound, interval partitioning and polyhedral approximations; see also the tutorial [15]. The L-shaped method has been extended to accommodate for integer variables with linear constraints in the second stage, when the first-stage problem is a pure binary problem [16], or, more in general, by incorporating dual cuts based on integer programming duality [4]. An algorithm based on outer approximation cuts [7] is discussed in [18] for the case in which the second-stage problems include convex nonlinear constraints. A very general approach, that can in principle deal with mixed-integer as well as nonlinear problems, is the stochastic branch-and-bound algorithm presented in [22]; the main obstacle to its implementation is the design of efficient upper and lower bounding procedures for the subproblems created by branching, and this ultimately determines the effectiveness of the algorithm.

In this paper we address the solution of problem (CCP) when the first-stage problem is a (potentially mixed-integer) linear program, and the second-stage problems are convex mixed-integer nonlinear programs (i.e., the sets $C_{x,y}(\omega)$ are described by convex constraints and integrality requirements on some or all of the variables). This very general class of problems encompasses all the ones discussed in the previous paragraph and is considered to be very difficult to solve in practice. The main ingredient of our algorithm is a sequential convexification procedure that generates valid inequalities for the convex hull of the second-stage problems, at the cost of solving several (small) mixed-integer programs. These inequalities are integrated within a branch-and-cut framework that acts in the space of the x variables, as well as additional binary variables used to determine which scenarios are feasible to satisfy the chance constraint. Branching is undertaken to ensure integrality of the x variables, if any, and feasibility for the second-stage problems, which may require spatial branching, in general. We show that the proposed algorithm converges in finite time to a point that is optimal for a relaxation of the original problem, where the relaxation amount can be chosen arbitrarily close to zero. We propose several practical enhancements of the algorithm to increase its computational performance: primal heuristics, warm-starts of the convexification procedure, and solution of the deterministic equivalent of partially-fixed branch-and-bound nodes to close them immediately. We show that all these components contribute to the effectiveness of our implementation, and allow us to solve instances of a mixed-integer two-stage chance-constrained resource allocation problem, with hundreds of scenarios, that are out of reach for a commercial solver applied to the deterministic equivalent formulation.

The rest of this paper is organized as follows. In Section 2 we formally introduce the considered problem, present the general scheme of our decomposition approach, and give results about its finite convergence. Section 3 gives some algorithmic details and enhancements, while Section 4 presents the outcome of a comprehensive computational analysis of the performances of the algorithm on two stochastic problems. Finally, we draw some conclusions in Section 5.

2. Decomposition algorithm for (CCP). Recall that our goal is to solve problem (CCP), restated here for convenience:

$$\text{(CCP)} \qquad \min\{cx : \Pr(x \in C_x(\omega)) \geq 1 - \alpha, x \in X\}.$$

We denote by Proj_x the projection of a set onto the space of the x variables, by Conv the convex hull, and by Cont the continuous relaxation (i.e., the set obtained by relaxing any integrality

requirements on the variables defining the set). Given $S \subset \mathbb{R}^n$ and $\epsilon \geq 0$, we denote $S + \epsilon := \{x \in \mathbb{R}^n : \|x - y\| \leq \epsilon \text{ for some } y \in S\}$, where we use $\|\cdot\|$ to indicate ℓ_2-norm. To obtain a deterministic reformulation for (CCP), we make the following assumptions:

1. the sample space Ω is discrete and finite, and in particular $\Omega = \{\omega^k : k = 1, \ldots, h\}$;
2. $C_x(\omega^k) = \text{Proj}_x(C_{x,y}(\omega^k))$, where $C_{x,y}(\omega^k) = \{(x, y) : x \in \mathbb{R}^{n_x}, y \in \mathbb{R}^{n_y(\omega^k)}, g^k(x, y) \leq 0, y_j \in Y^k\}$, $Y^k = \{y \in \mathbb{R}^{n_y(\omega^k)} : y_j \in \mathbb{Z}\ \forall j \in \mathcal{I}_k\}$, and $g^k(x, y)$ is a vector of convex functions $(g^k_1, \ldots, g^k_m)$;
3. X is closed and $X \subseteq [-U, U]^{n_x}$ for some constant U.

We discussed the first assumption in the introduction. The second assumption states that the feasible set for each scenario is a mixed-integer convex set. The motivation for the third assumption will be apparent shortly. Using the first assumption, and denoting $p_k = \Pr(\omega = \omega^k)$, we introduce a set of indicator variables z_k and rewrite the problem as

$$\text{(CCP-R)} \qquad \begin{array}{lrcl} \min & cx & & \\ \text{s.t.:} & x & \in & X \\ k = 1, \ldots, h & z_k = 0 \Rightarrow x & \in & C_x(\omega^k) \\ & \sum_{k=1}^h p_k z_k & \leq & \alpha \\ k = 1, \ldots, h & z_k & \in & \{0, 1\}. \end{array}$$

Using the second and third assumption, the problem can be modeled as the mixed-integer nonlinear program (MINLP)

$$\text{(CCP-MINLP)} \qquad \begin{array}{llllllll} \min & cx & & & & & & \\ \text{s.t.:} & Ax & & & & & \geq & b \\ & g^1(x, & y^1) & & & & \leq & M^1 z_1 \\ & \vdots & & \ddots & & & \vdots & \\ & g^h(x, & & & & y^h) & \leq & M^h z_h \\ & & p_1 z_1 + & \ldots & + & p_h z_h & \leq & \alpha \\ & & y^1 & & & & \in & Y^1 \\ & & & \ldots & & & & \\ & & & & & y^h & \in & Y^h \\ & & z_1, & \ldots & & z_h & \in & \{0, 1\} \end{array}$$

In this formulation, we assume that M^k are vectors of constants large enough to deactivate the corresponding constraints if $z_k = 1$. Such constants exist thanks to the third assumption. Since there is no objective function contribution associated with variables y, the second stage problems are feasibility problems; alternative models with second-stage objective function contributions can be considered, e.g., by enlarging the vector of first-stage variables, developing specialized optimality cuts, or even introducing a "recovery mode" for violated scenarios, see [17]. One motivation for the third assumption can now be properly explained: it ensures that the recession cone of (CCP-MINLP) and the one of (CCP-R) are the same, as they coincide with the set $\{0\}$. The assumption that the recession cones are the same is necessary for mixed-integer representability [10]. A further motivation is technical: we need the third assumption for our convergence proof. We remark that, although we discuss the case in which all x variables are continuous, our approach is based on a branch-and-cut framework in the x-space, therefore from a theoretical point of view it can be easily extended to handle integrality requirements on some x variables. The implementation of the algorithm would however be more complex, therefore we do not explicitly consider this case.

Formulation (CCP-MINLP) is the deterministic equivalent of (CCP), under the three assumptions stated above. It can be solved with a convex MINLP solver, or, in case the constraints $g^1, \dots, g^h$ are linear, with a MILP solver. However, such formulation has two main drawbacks: its size, and its weak continuous relaxation. The size is an issue because (CCP-MINLP) includes all variables and constraints for the second-stage problems: since the problem is then solved via branch-and-cut, increasing the size linearly with the number of scenarios generally leads to exponential growth of the running time in practice. The weak continuous relaxation is due to the presence of many big-M constraints, which can lead to large gaps between the original problem and its relaxation.

Several decomposition algorithms for problems with a structure similar to (CCP-MINLP) have been proposed, see also the discussion in Section 1. The algorithm that we propose is inspired, most notably, by [19] and [18], both of which employ a similar branch-and-bound scheme, and generate valid inequalities for second-stage feasible sets $C_{x,y}(\omega)$ that are linear and nonlinear convex, respectively. We follow a decomposition approach whereby we define a master problem with first-stage variables x only, and h subproblems, one for each scenario, involving the respective scenario-dependent constraints. The master problem is a relaxation of the original problem, as all the constraints depending on second-stage variables have been eliminated. Therefore, whenever the solution of the master problem does not satisfy the constraints of some active scenario ($z_k = 0$), we want to generate cuts for the corresponding set $C_{x,y}(\omega^k)$, and add them to the master problem. This is the same scheme followed in [19, 18], but the task in this paper is complicated by the integrality restrictions on the second-stage variables $y^1, \dots, y^h$, so that the generation of valid inequalities for $C_{x,y}(\omega^k)$ is considerably more involved. Although we add these cuts as big-M constraints, they only involve x variables, therefore we need smaller values of the big-M coefficients than in CCP-MINLP.

To develop intuition, let us consider the case in which we want all scenarios to be satisfied, i.e., (CCP-MINLP) with $\alpha = 1$. At a high level, the problem that we need to solve can then be stated in terms of the following simpler question: given a point $\hat{x}$ from the master problem and a scenario index k, is $\hat{x}$ feasible for scenario ω^k, i.e., does there exist $\hat{y}$ such that $(\hat{x}, \hat{y}) \in C_{x,y}(\omega^k)$, and if not, can we somehow exclude $\hat{x}$ from the master? A systematic procedure for this task can be used to solve the original problem: the master is a relaxation of the original problem, and with the above procedure we could exclude every candidate solution until we find one that is feasible for all $C_{x,y}(\omega^k)$, $k = 1, \dots, h$ (if we do not require all scenarios to be satisfied, i.e., $\alpha < 1$, then the algorithm must be modified to eliminate only those $\hat{x}$ that are not feasible for a number of scenarios sufficient to satisfy the chance constraint). Notice that if $\hat{x}$ is infeasible for scenario ω^k, the nonconvexity of $C_{x,y}(\omega^k)$ (due to integrality) may prevent us from deriving a valid linear inequality that cuts off $\hat{x}$, so cutting may not be sufficient by itself. We now discuss in more detail how to eliminate $\hat{x}$ when it is not feasible for a scenario ω^k.

By definition of the second-stage feasible sets $C_{x,y}(\omega^k)$, a point $\hat{x}$ is feasible for $C_{x,y}(\omega^k)$ if and only if $\hat{x} \in \text{Proj}_x(C_{x,y}(\omega^k))$. For simplicity, we consider a fixed scenario and therefore temporarily drop the symbol ω^k: the discussion applies in the same way to all scenarios. There are three notable cases in which $\hat{x}$ is not feasible and we want to exclude it, leading to three different procedures of increasing computational cost:

1. $\hat{x} \notin \text{Proj}_x\text{Cont}(C_{x,y})$;
2. $\hat{x} \in \text{Proj}_x\text{Cont}(C_{x,y})$ and $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y})$;
3. $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$.

In the first case, it is sufficient to find an inequality valid for the projection of the continuous relaxation of $C_{x,y}$; this can be done with standard techniques, such as Benders or generalized

Benders [9] cuts for linear and nonlinear convex sets, respectively. We use the outer approximation procedure of [18]. In the second case, we need a valid inequality for $\text{Proj}_x\text{Conv}(C_{x,y})$, which in turn requires knowledge of $\text{Conv}(C_{x,y})$; note that such an inequality exists, because we are separating from a convex set. We will generate a valid inequality with a sequential convexification procedure described subsequently. In the last case, a separating inequality does not exist, therefore to separate $\hat{x}$ we perform spatial branching.

The pseudo-code of our decomposition approach is provided in Algorithm 2.1. Since the master problem involves the x variables only, the separation routines must find a cut in the x space. In the remainder of this section, we provide the separation algorithms for cases 1 and 2 and we specify how branching is performed when case 3 occurs.

2.1. Case 1: separation when $\hat{x} \notin \text{Proj}_x\text{Cont}(C_{x,y})$. In order to check whether $\hat{x} \in \text{Proj}_x\text{Cont}(C_{x,y})$ and, when this is not the case, determine a cut to separate $\hat{x}$, let us define the problem

$$\min_{x \in \text{Proj}_x\text{Cont}(C_{x,y})} \frac{1}{2}\|x - \hat{x}\|^2. \tag{PROJ}$$

It it clear that the optimal solution value of (PROJ) is strictly positive if and only if $\hat{x} \notin \text{Proj}_x\text{Cont}(C_{x,y})$. In this case, Theorem 1 of [18], adapted to our case, allows us to compute a valid cut to separate $\hat{x}$:

THEOREM 2.1. *Let $\text{Cont}(C_{x,y})$ be a closed set such that $\text{Proj}_x\text{Cont}(C_{x,y})$ is convex, and $\hat{x} \notin \text{Proj}_x\text{Cont}(C_{x,y})$. Let $\bar{x}^*$ be the optimal solution to (PROJ) with positive objective function value. Then, the hyperplane*

$$(\hat{x} - \bar{x}^*)^T(x - \bar{x}^*) \leq 0$$

separates $\hat{x}$ from $\text{Proj}_x\text{Cont}(C_{x,y})$. This hyperplane is the deepest valid cut that separates $\hat{x}$ from $\text{Proj}_x\text{Cont}(C_{x,y})$, if depth is computed in ℓ_2-norm.

We refer to [18] for a proof of this result.

2.2. Case 2: separation when $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y})$. Let us assume that $\hat{x} \in \text{Proj}_x\text{Cont}(C_{x,y})$, because otherwise we would fall under Case 1. Our approach for Case 2 is (i) to check if $\hat{x}$ lies in $\text{Proj}_x\text{Conv}(C_{x,y})$ and, if not, (ii) to define a supporting hyperplane for $\text{Proj}_x\text{Conv}(C_{x,y})$ separating $\hat{x}$. We show that (i) and (ii) can be performed by solving an MINLP with nonconvex continuous relaxation. As this is computationally difficult, we introduce an iterative procedure for the MINLP that solves two more tractable sub-problems: a (continuous) nonlinear problem, and an MINLP with convex continuous relaxation. We discuss the convergence of this procedure, showing finite convergence in the linear case and asymptotic convergence in the nonlinear convex case. For the latter case, after introducing proper numerical tolerances, we show that we can verify (i) and derive a cut (ii) with any desired precision in finite time.

Consider the following problem:

$$\begin{array}{rrcl} \min & \|\hat{x} - \sum_{i=1}^{n_x+1} \lambda_i x^i\| & & \\ \text{s.t.:} & \sum_{i=1}^{n_x+1} \lambda_i & = & 1 \\ i = 1, \ldots, n_x + 1 & \lambda_i & \geq & 0 \\ i = 1, \ldots, n_x + 1 & x^i, y^i & \in & C_{x,y}. \end{array} \tag{DIST}$$

We distinguish two possible outcomes:

Algorithm 2.1 Decomposition Algorithm

1: Define a master problem P^0 as

(MASTER)
$$\begin{array}{lrcl} \min & cx & & \\ \text{s.t.:} & Ax & \geq & b \\ & \sum_{k=1}^{h} p_k z_k & \leq & \alpha \\ & z & \in & \{0,1\}^h; \end{array}$$

2: Initialize the list Q of active nodes with MASTER;
3: **repeat**
4: Select a subproblem from Q;
5: Solve the continuous relaxation and let $(\hat{x}, \hat{z})$ be an optimal solution;
6: **if** subproblem is infeasible or pruned by bound **then**
7: **goto** 3;
8: **end if**
9: **if** $\exists k \in \{1, \ldots, h\}$ such that $0 < \hat{z}_k < 1$ **then**
10: branch on variable z_k: define two subproblems and add them to Q;
11: **goto** 3;
12: **end if**
13: **for each** $k \in \{1, \ldots, h\} : \hat{z}_k = 0$ **do**
14: **if** $\nexists x \in \text{Proj}_x\text{Cont}(C_{x,y}(\omega^k)) : \|x - \hat{x}\| \leq \epsilon$ **then** ▷ Case 1
15: Separate $\hat{x}$ from $\text{Proj}_x\text{Cont}(C_{x,y}(\omega^k))$ with an inequality $\gamma x \geq \beta$;
16: Add inequality $\gamma x \geq \beta - M z_k$ to MASTER;
17: **goto** 5;
18: **end if**
19: **end for**
20: **for each** $k \in \{1, \ldots, h\} : \hat{z}_k = 0$ **do**
21: **if** $\nexists x \in \text{Proj}_x\text{Conv}(C_{x,y}(\omega^k)) : \|x - \hat{x}\| \leq \epsilon$ **then** ▷ Case 2
22: Separate $\hat{x}$ from $\text{Proj}_x\text{Conv}(C_{x,y}(\omega^k))$ with an inequality $\gamma x \geq \beta$;
23: Add inequality $\gamma x \geq \beta - M z_k$ to MASTER;
24: **goto** 5;
25: **end if**
26: **end for**
27: **if** $\hat{x}$ is not feasible for some active scenario **then** ▷ Case 3
28: branch on a variable x_j: define two subproblems and insert them into Q;
29: **goto** 3;
30: **end if**
31: update the incumbent;
32: **until** list Q is empty

1. The optimal objective function value is equal to zero; thus, there exist $n_x + 1$ points x^i in $\text{Proj}_x C_{x,y}$ whose convex combination is equal to $\hat{x}$, implying that $\hat{x}$ lies in $\text{Proj}_x\text{Conv}(C_{x,y})$.
2. The optimal objective function value is larger than zero; in this case $\hat{x}$ does not lie in $\text{Proj}_x\text{Conv}(C_{x,y})$, because if it did, by Carathéodory's theorem, there would exist $n_x + 1$ points x^i and multipliers λ_i to fall into the previous case.

Note that in problem DIST both x^i and λ_i are variables, so the formulation, in addition to its large size, has in general a nonconvex continuous relaxation; thus, it is very difficult to solve in practice, and may be as hard as the original problem (CCP).

We therefore introduce an iterative procedure that converges to an optimal solution of DIST. The procedure is initialized with the feasible point in $C_{x,y}$ closest to $\hat{x}$, obtained by solving the following convex MINLP:

$$\text{(FINDX1)} \qquad x^1 \leftarrow \arg_x \min_{(x,y)\in C_{x,y}} \|\hat{x} - x\|.$$

If the distance between this point and $\hat{x}$ is 0 (or lower than a fixed tolerance ϵ, see the discussion surrounding Algorithm 2.3), $\hat{x}$ is feasible (or we consider it feasible within the tolerance) and the procedure stops; no cut is generated. Otherwise, an iterative process starts where, at each iteration, we aim to find a new point in $C_{x,y}$ that, in a convex combination with all previously found points, decreases the distance from $\hat{x}$.

For this purpose, we define two new problems. Let S be a set of points in $\text{Proj}_x C_{x,y}$; initially, $S = \{x^1\}$. The first problem minimizes the distance from the convex combination of the points in the current set S to $\hat{x}$, and is defined as follows:

$$\text{(MINDIST)} \qquad \begin{array}{rrcl} \min & \|\hat{x} - \sum_{i=1}^{|S|} \lambda_i x^i\| & & \\ \text{s.t.:} & \sum_{i=1}^{|S|} \lambda_i & = & 1 \\ i = 1, \dots, |S| & \lambda_i & \geq & 0. \end{array}$$

In this model, a continuous variable λ_i is associated with each point $x^i \in S$; note that the x^i here are data. Given an optimal solution $\bar{\lambda}_i$ of (MINDIST), let us define $\bar{x} = \sum_{i=1}^{|S|} \bar{\lambda}_i x^i$. If the distance between $\hat{x}$ and $\bar{x}$ is 0, then $\hat{x} \in \text{Proj}_x \text{Conv}(C_{x,y})$, and the procedure stops. Otherwise, we define a second problem to determine a new point $x^{|S|+1}$ to be added to S, with the goal of decreasing the objective function value of (MINDIST). This is equivalent to identifying a descent direction from $\bar{x}$ that decreases the distance to $\hat{x}$, as follows:

$$\text{(NEWPOINT)} \qquad \begin{array}{rrcl} \max & (\hat{x} - \bar{x})^T (x^{|S|+1} - \bar{x}) & & \\ \text{s.t.:} & (x^{|S|+1}, y) & \in & C_{x,y}. \end{array}$$

Here, the only (vector-valued) decision variable is the point $x^{|S|+1}$. As shown below, if the optimal solution value of this problem is larger than zero, adding the new point $x^{|S|+1}$ to S decreases the objective function value of (MINDIST). Otherwise, the iterative procedure stops, returning the point $\bar{x}^*$. If the distance between $\hat{x}$ and $\bar{x}^*$ is still strictly positive, we can conclude that $\hat{x} \notin \text{Proj}_x \text{Conv}(C_{x,y})$, finding an inequality to separate $\hat{x}$. The inequality is the hyperplane orthogonal to $\hat{x}$ at $\bar{x}^* = \sum_{i=1}^{|S|} \lambda_i x^i$. The entire procedure is described in Algorithm 2.2. For now, let us assume that we are working with infinite precision, i.e., all calculations can be carried out exactly.

PROPOSITION 2.2. *The sequence $\bar{x}$ of points generated by Algorithm 2.2 as solutions to the problem (MINDIST) converges to $\hat{x}$ or to a point $\bar{x}^* = \sum_{i=1}^{|S|} \lambda_i^* x^i \in \text{Proj}_x \text{Conv}(C_{x,y})$ that has minimum distance from $\hat{x}$. An optimal solution to (DIST) is given by $\lambda_i^* > 0$ and the associated $x^i \in S$.*

Proof. We claim that the objective value of (NEWPOINT) is strictly positive if and only if the objective value of (MINDIST) decreases in the following iteration. To show this, let us consider a set of feasible points S and the corresponding $\bar{x}$, i.e., the point in the convex hull of S that is closets to $\hat{x}$. The hyperplane $(\hat{x} - \bar{x})^T (x - \bar{x}) = 0$ is supporting to the convex hull of S at $\bar{x}$, and it defines the halfspace $H_1 = \{x : (\hat{x} - \bar{x})^T (x - \bar{x}) \leq 0\}$ and its complement $H_2 = \{x : (\hat{x} - \bar{x})^T (x - \bar{x}) > 0\}$. Note that $S \subset H_1$. Consider a new point x' for which

Algorithm 2.2 ProjectAndCut$(\hat{x}, C_{x,y})$

$x^1 \leftarrow \arg_x \min_{(x,y)\in C_{x,y}} \|x - \hat{x}\|$
if $\|\hat{x} - x^1\| = 0$ **then**
  return "$\hat{x}$ is feasible for $C_{x,y}$"
end if
$S \leftarrow \{x^1\}$
$d_1 \leftarrow \|\hat{x} - x^1\|$
$\bar{x} \leftarrow x^1$
repeat
  Solve (NEWPOINT); let $x^{|S|+1}$ be its optimal solution
  $S \leftarrow S \cup \{x^{|S|+1}\}$
  Solve (MINDIST); let $d_{|S|}$ be its optimal value and $\bar{\lambda}$ the optimal solution
  $\bar{x} \leftarrow \sum_{i=1}^{|S|} \bar{\lambda}_i x^i$
until ($d_{|S|} = 0$ **or** $d_{|S|} = d_{|S|-1}$)
$\bar{x}^* \leftarrow \bar{x}$
if $d_{|S|} = 0$ **then**
  return "$\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$"
else
  return valid inequality $(\hat{x} - \bar{x}^*)^\top (x - \bar{x}^*) \leq 0$
end if

$(\hat{x} - \bar{x})^T(x' - \bar{x}) \leq 0$: solving (MINDIST) with the set of points $S \cup \{x'\}$ would still yield $\bar{x}$ as the optimal solution, showing that the objective value of (NEWPOINT) must be strictly positive to decrease (MINDIST) at subsequent iterations. Now we show that when the objective function value of (NEWPOINT) is strictly positive, the value of (MINDIST) decreases at the following iteration. Consider a new point x' for which $(\hat{x} - \bar{x})^T(x' - \bar{x}) > 0$. Since $x' \in H_2$, the projection of the segment $x' - \bar{x}$ onto $\hat{x} - \bar{x}$ points toward $\hat{x}$, implying that it is a descent direction for (MINDIST). Furthermore, the whole segment $x' - \bar{x}$ belongs to the convex hull of $S \cup \{x'\}$, therefore we can reduce the objective value of (MINDIST) at the following iteration. This concludes the proof of the claim.

Thus, during the iterations of the *repeat-until* loop of Algorithm 2.2, the solution values $d_{|S|}$ of (MINDIST) define a monotonically decreasing sequence that is lower bounded by 0, which must converge to its infimum. If the infimum is 0, then the sequence of points generated by the algorithm converges to $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$. If the infimum is $d > 0$, the corresponding solutions to (MINDIST) converge to a point $\bar{x}^*$ with distance d from $\hat{x}$. Assume that $\bar{x}^*$ is not the closest point to $\hat{x}$. But then there is a descent direction for (MINDIST), the objective value of (NEWPOINT) is strictly positive, and the objective value of (MINDIST) could be further reduced, contradicting the fact that d is the infimum of the sequence of values $d_{|S|}$. □

Proposition 2.2 shows that Algorithm 2.2 converges to an optimal solution for problem (DIST). However, the following negative result shows that convergence may be obtained only asymptotically in case $C_{x,y}$ is a mixed-integer nonlinear (convex) set.

PROPOSITION 2.3. *There exists a mixed-integer set $C_{x,y} \subset \mathbb{R}^2$, with a convex continuous relaxation* $\text{Cont}(C_{x,y})$*, a point $\hat{x}$ and an initial point x^1, such that for any finite number of steps, Algorithm 2.2 does not determine the optimal solution of* (DIST).

Proof. We construct such an example in Fig. 1, where we show two circles, representing the

feasible set, and the (sub)sequence of feasible points x^1, x^2, x^3 and x^4 returned by (NEWPOINT). We give a geometric sketch of the proof for simplicity: a detailed proof can be derived from this sketch, but would be considerably longer. At every iteration, the facet of the convex hull of S generated by Algorithm 2.2 is not perfectly horizontal; as a consequence, the vector from the solution $\bar{x}$ of (MINDIST) to $\hat{x}$ is not vertical. Hence, at every iteration the next feasible point generated by (NEWPOINT) lies on the outer arc of one of the two circles (outside the dashed lines), and the sequence of generated points does not reach the top of the circles in finite time. However, the optimal solution to (DIST) can only be obtained when both points at the top of the two circles belong to S. ∎

The previous result shows that in some cases we may not be able to generate the (exact) convex hull in finite time. We next show that the negative result of Proposition 2.3 does not apply in the relevant special case in which $C_{x,y}$ is a mixed-integer linear set; indeed, in this case convergence occurs in a finite number of iterations. Subsequently, we show that an approximate notion of convergence can be attained in finite time even in the nonlinear case.

PROPOSITION 2.4. *Suppose $C_{x,y}$ is a mixed-integer linear set. Then, Algorithm 2.2 terminates after a finite number of steps.*

Proof. Since (NEWPOINT) has linear objective, line 9 always returns an extreme point of $C_{x,y}$. As the number of extreme points is finite, after a finite number of iterations set S contains all extreme points of $C_{x,y}$. Therefore we are able to generate the convex hull of $C_{x,y}$ and Algorithm 2.2 necessarily terminates. ∎

Remark 2.5. When $C_{x,y}$ is a mixed-integer linear set, Algorithm 2.2 may need to generate, in the worst case, all the extreme points of $C_{x,y}$ before convergence. To see this, consider the problem instance discussed in the proof of Prop. 2.3. Let $x^1, x^2, \ldots,$ be the infinite sequence of points generated by Algorithm 2.2. Let t be any finite index and truncate to sequence at x^t. Partition these points into S_1 and S_2, where S_1 includes all points that belong to the left feasible set in Fig. 1, and S_2 all points that belong to the right feasible set in Fig. 1. Then, consider the set $\tilde{C}_{x,y} = \text{Conv}(S_1) \cup \text{Conv}(S_2)$. It is easy to see that Algorithm 2.2, when applied to $\tilde{C}_{x,y}$, would generate the same sequence of points before termination, and therefore it would generate all the extreme points of $\tilde{C}_{x,y}$.

As mentioned, despite the negative result of Proposition 2.3, Algorithm 2.2 converges in a finite number of iterations if some numerical tolerances are introduced. The resulting approximate procedure, detailed in Algorithm 2.3, returns a point $\bar{x}$ that is close (although possibly not the closest) to $\hat{x}$. Note that, as opposed to Algorithm 2.2, for convenience we now assume that the initial point x^1 is given as an input.

PROPOSITION 2.6. *Given two values of tolerance $\epsilon > 0$ and $\zeta > 0$, Algorithm 2.3 terminates in a finite number of iterations.*

Proof. At each iteration of Algorithm 2.3, either the distance of $\bar{x}$ from $\hat{x}$ is reduced by a value larger than ζ, or it is not. The former case can only happens a finite number of times, because the distance between $\hat{x}$ and $\text{Proj}_x\text{Conv}(C_{x,y})$ is finite; in the latter case, the algorithm stops. Thus, the algorithm runs for a finite number of steps. ∎

When Algorithm 2.3 stops with $d_{|S|} > \epsilon$, not only the computed point $\bar{x}$ may be in the strict interior of $\text{Proj}_x\text{Conv}(C_{x,y})$, but we also have no upper bound on the distance of $\bar{x}$ from the boundary of $\text{Proj}_x\text{Conv}(C_{x,y})$. Hence, the inequality $(\hat{x} - \bar{x})^\top (x - \bar{x}) \leq 0$ may not be valid (i.e.,

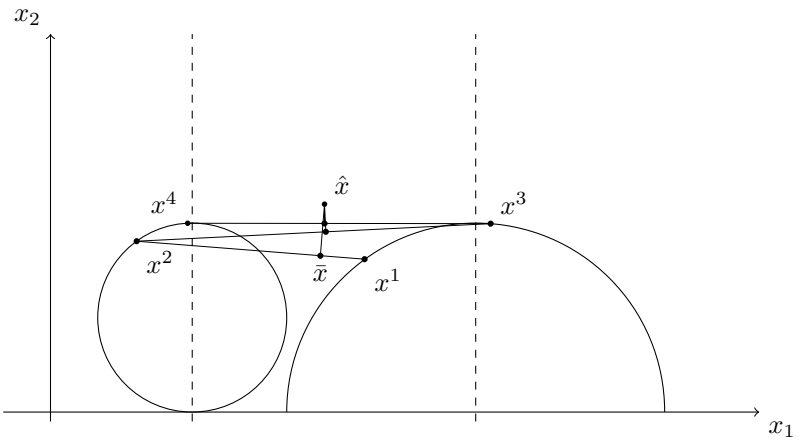

FIG. 1. *Two-dimensional example showing that Algorithm 2.2 may not converge in a finite number of steps, see Prop. 2.3.*

Algorithm 2.3 ProjectAndCutWithTolerance$(\hat{x}, x^1, C_{x,y}, \epsilon, \zeta)$

$\bar{x} \leftarrow x^1$
$S \leftarrow \{x^1\}$
$d_1 \leftarrow \|\hat{x} - x^1\|$
repeat
  Solve (NEWPOINT); let $x^{|S|+1}$ be its optimal solution
  $S \leftarrow S \cup \{x^{|S|+1}\}$
  Solve (MINDIST); let $d_{|S|}$ be its optimal value and $\bar{\lambda}$ the optimal solution
  $\bar{x} \leftarrow \sum_{i=1}^{|S|} \bar{\lambda}_i x^i$
until ($d_{|S|} \leq \epsilon$ **or** $d_{|S|-1} - d_{|S|} \leq \zeta$)
return $\bar{x}$

it may cut some feasible solution), and needs to be relaxed before it can be added to (MASTER) (line 21 of Algorithm 2.1). We relax the inequality by increasing its r.h.s. to a value that makes it supporting to Conv$(C_{x,y})$. However, if the tolerances ϵ and ζ are large, the relaxed cut may not cut $\hat{x}$ off, as shown in the left part of Figure 2, where the dashed line represents the "ideal" cut (associated with $\bar{x}^*$), the dotted line is the invalid cut (through $\bar{x}$), and the solid line corresponds to its valid relaxed counterpart. The right part of the figure shows that reducing the numerical tolerances, which leads to performing additional iterations of the repeat-until loop of Algorithm 2.3, allows convergence to a point closer to $\bar{x}^*$ than in the previous case: this yields a relaxed cut that now separates $\hat{x}$.

Hence, in case the relaxed cut does not separate $\hat{x}$, our approach is to reduce the tolerance ζ (that controls the repeat-until loop), and execute Algorithm 2.3 with the new tolerance. The resulting scheme, described in Algorithm 2.4, generates a sequence of points $\bar{x}^j$, until we get to a point that allows us to compute a valid inequality to separate $\hat{x}$, or to conclude that $\hat{x}$ is optimal within an ϵ tolerance. The next proposition shows that Algorithm 2.4 always terminates in a finite number of iterations.

PROPOSITION 2.7. *Given a value of tolerance $\epsilon > 0$ and an initial value $\zeta^0 > 0$, Algorithm 2.4*

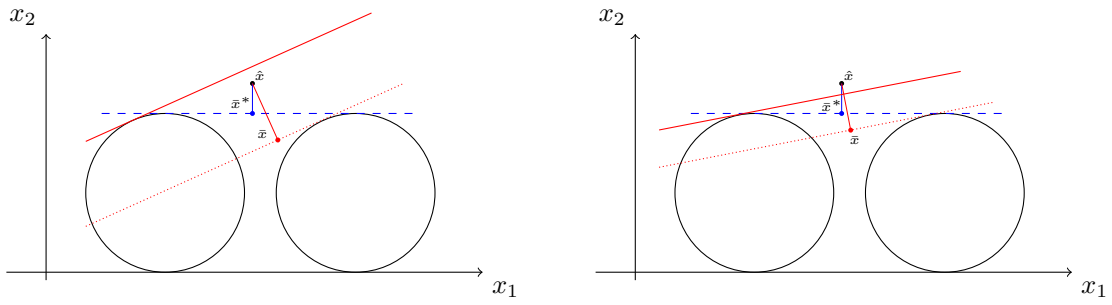

FIG. 2. *Early termination of Algorithm* **2.4 RelaxAndIterate** *produces an invalid cut, whose relaxation is unable to cut $\hat{x}$ (left). Reducing tolerance ζ yields a relaxed cut that separates $\hat{x}$ (right).*

Algorithm 2.4 RelaxAndIterate($\hat{x}, C_{x,y}, \epsilon, \zeta^0$)

1: $x^1 \leftarrow \arg\min_{(x,y)\in C_{x,y}} \|x - \hat{x}\|$
2: **if** $\|\hat{x} - x^1\| \leq \epsilon$ **then**
3: **return** "$\hat{x}$ is ϵ-feasible for $C_{x,y}$"
4: **end if**
5: $j \leftarrow 0$
6: **repeat**
7: $\bar{x}^j \leftarrow$ ProjectAndCutWithTolerance($\hat{x}, x^1, C_{x,y}, \epsilon, \zeta^j$)
8: **if** $\|\hat{x} - \bar{x}^j\| \leq \epsilon$ **then**
9: **return** "$\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y}) + \epsilon$"
10: **end if**
11: $H \leftarrow \text{relax}((\hat{x} - \bar{x}^j)^\top (x - \bar{x}^j) \leq 0)$
12: choose $\zeta^{j+1} < \zeta^j$
13: $j \leftarrow j + 1$
14: **until** H separates $\hat{x}$
15: **return** H

terminates in a finite number of iterations by either returning a valid inequality to separate $\hat{x}$, or "$\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y}) + \epsilon$", or "$\hat{x}$ is ϵ-feasible for $C_{x,y}$".

Proof. Proposition 2.2 states that, if tolerances ϵ and ζ were set to 0, the sequence $\bar{x}^j$ generated at step 4 of Algorithm 2.4 would converge to either i) $\hat{x}$, when $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$, or ii) a point, say $\bar{x}^*$, on the boundary of $\text{Proj}_x\text{Conv}(C_{x,y})$.

Assume now that the tolerances are set to strictly positive values. In case i) we have that for each $\epsilon > 0$ there is a finite index $\bar{\jmath}$ such that $\|\hat{x} - \bar{x}^j\| < \epsilon$ for $j \geq \bar{\jmath}$; the algorithm reaches $\bar{\jmath}$ in a finite number of iterations, and then returns "$\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y}) + \epsilon$". In case ii), when $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y})$, we have that for every $\delta > 0$, there exists a finite index $\bar{\jmath}$ such that $\|\bar{x}^* - \bar{x}^j\| < \delta$ for $j \geq \bar{\jmath}$. Now, we distinguish two sub-cases of ii): when $0 < \|\hat{x} - \bar{x}^*\| < \epsilon$, for small enough $\delta > 0$, we have $\|\hat{x} - \bar{x}^j\| < \epsilon$ for $j \geq \bar{\jmath}$, and the algorithm returns "$\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y}) + \epsilon$". When $\|\hat{x} - \bar{x}^*\| > \epsilon$, we claim (see Prop. 2.8) that there exists a small enough $\delta > 0$ such that if $\|\bar{x}^* - \bar{x}^j\| < \delta$, a (not valid) inequality $(\hat{x} - \bar{x}^j)^\top (x - \bar{x}^j) \leq 0$ can be relaxed to a valid inequality that separates $\hat{x}$. This inequality is obtained in finite time, i.e., we can compute it at iteration $\bar{\jmath}$. □

It still remains to show that for $\bar{x}^j$ close enough to $\bar{x}^*$, we can relax the inequality returned by

Algorithm 2.4 and separate $\hat{x}$. This is our next result.

PROPOSITION 2.8. *For every $\epsilon > 0$ and $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y}) + \epsilon$, there exists $\delta > 0$ such that if $\|\bar{x}^{\bar{\jmath}} - \bar{x}^*\| < \delta$, the inequality $(\hat{x} - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}) \leq 0$ can be relaxed to be valid for $\text{Proj}_x\text{Conv}(C_{x,y})$ while still cutting off $\hat{x}$. The relaxation amount is upper bounded by ϵ^2.*

Proof. Notice that by convexity and the fact that $\bar{x}^* \in \text{Proj}_x\text{Conv}(C_{x,y})$, we have

$$\max_{x \in \text{Proj}_x\text{Conv}(C_{x,y})} (\hat{x} - \bar{x}^*)^\top (x - \bar{x}^*) = 0. \tag{2.1}$$

To find the tightest relaxation that makes the inequality $(\hat{x} - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}) \leq 0$ valid, we compute

$$b := \max_{x \in \text{Proj}_x\text{Conv}(C_{x,y})} (\hat{x} - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}) = \max_{(x,y) \in C_{x,y}} (\hat{x} - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}), \tag{2.2}$$

obtaining the inequality $(\hat{x} - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}) \leq b$. Now, let us expand (2.2) as

$$\begin{aligned}\max_{(x,y) \in C_{x,y}} (\hat{x} - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}) = \max_{(x,y) \in C_{x,y}} (\hat{x} - \bar{x}^* + \bar{x}^* - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^* + \bar{x}^* - \bar{x}^{\bar{\jmath}}) \leq \\ \max_{(x,y) \in C_{x,y}} (\hat{x} - \bar{x}^*)^\top (\bar{x}^* - \bar{x}^{\bar{\jmath}}) + (\bar{x}^* - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}),\end{aligned}$$

where we used (2.1). Since $\|(\hat{x} - \bar{x}^*)\| \leq 2U$ and $\|(x - \bar{x}^{\bar{\jmath}})\| \leq 2U$, and $\|(\bar{x}^* - \bar{x}^{\bar{\jmath}})\| < \delta$ by assumption, we can upper bound the above expression by $4U\delta$. Thus, as long as $\delta \leq \frac{\epsilon^2}{4U}$, $b < \epsilon^2$, we have

$$(\hat{x} - \bar{x}^{\bar{\jmath}})^\top (\hat{x} - \bar{x}^{\bar{\jmath}}) = \|\hat{x} - \bar{x}^{\bar{\jmath}}\|^2 \geq \epsilon^2 > b,$$

showing that the relaxed inequality cuts off $\hat{x}$. ∎

2.3. Case 3: $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$: spatial branching on variable x. When $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$ there is no way to separate this point through a valid cut expressed in terms of the first-stage variables x only. Hence, in this case we perform spatial branching on (MASTER), by selecting a variable x_j that is neither at its lower nor at its upper bound in $\hat{x}$, and splitting the feasible set by imposing $x_j \leq \hat{x}_j$ and $x_j \geq \hat{x}_j$, respectively. As discussed in the proof of Theorem 2.9, whenever $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$, such a variable exists.

2.4. Convergence of the Decomposition Algorithm 2.1. We are now ready to state the main theoretical result of our paper, namely, the finite convergence of the Decomposition Algorithm 2.1 for any tolerance $\epsilon > 0$.

THEOREM 2.9. *Consider a problem of the form (CCP) satisfying Assumptions A1-A3, such that X is compact. Let $\epsilon > 0$. Let $\mathcal{C}$ be the cut generation procedure of Algorithm 2.4. Assume that the branching procedure is such that we always branch on a fractional variable z, if any, and on a first-stage variable x_j whose current value $\hat{x}_j$ is neither at its lower nor at its upper bound otherwise. Then, the algorithm terminates after a finite number of nodes and a finite number of calls to $\mathcal{C}$ with the following outcome: if the model*

$$\begin{array}{rrcl} \min & cx & & \\ s.t.: & x & \in & X \\ k = 1, \dots, h \quad z_k = 0 \Rightarrow & x & \in & C_x(\omega^k) + \epsilon \\ & \sum_{k=1}^h p_k z_k & \leq & \alpha \\ k = 1, \dots, h & z_k & \in & \{0, 1\}. \end{array} \tag{2.3}$$

does not admit a feasible solution, Algorithm 2.1 returns "infeasible". Otherwise, Algorithm 2.1 returns a point $(\hat{x^*}, \hat{z^*})$ *that is optimal for the above model* (2.3)*, i.e., a relaxed version of* (CCP) *where each second-stage feasible set is relaxed by* ϵ*.*

Proof. Since $z \in \{0,1\}^h$ and branching decisions (i.e., binary variables fixed at some value) are never relaxed in children nodes, we can branch on a z variable at most $O(2^h)$ times. Therefore, we now consider the behavior of the algorithm at some node N for a fixed vector $\hat{z}$.

We claim that after a finite number of calls to $\mathcal{C}$ with tolerance $\sqrt{\epsilon}$, we have $\hat{x} \in \text{Conv}(C_x(\omega^k)) + \epsilon \cap B$ for every $k : z_k = 0$, where B is the set defined by the branching decisions accumulated at node N. For any fixed k, we generalize the proof of [11] to allow for the cut relaxation step described in Algorithm 2.4; this requires some modifications in the proof. Define $G(x) := \min_{x' \in \text{Conv}(C_x(\omega^k)) \cap B} \|x - x'\|$, i.e., the distance between x and the set that we aim to converge to. Notice that with the cutting plane method, we are solving a sequence of problems of the form $\min_{x \in X, x \in R} c^\top x$ where R is a relaxation of $\text{Conv}(C_x(\omega^k)) \cap B$, and we iteratively add inequalities to R. Let t_i be the sequence of points generated by this algorithm, and assume that the i-th inequality is of the form

$$G(t_i) + \nabla G(t_i)^\top (x - t_i) \leq 0, \tag{2.4}$$

where $\nabla G(t_i)$ is a subgradient of G at t_i. By convexity of function G, this inequality is valid for the set $\{x : G(x) \leq 0\}$. Let $\bar{x}^{\mathbf{b}} := \arg\min_{x' \in \text{Conv}(C_x(\omega^k)) \cap B} \|t_i - x'\|$. Then, $G(t_i) = \|t_i - \bar{x}^{\mathbf{b}}\|$, and a subgradient of G at t_i is given by $\frac{t_i - \bar{x}^{\mathbf{b}}}{\|t_i - \bar{x}^{\mathbf{b}}\|}$. Writing $x - t_i$ as $x - \bar{x}^{\mathbf{b}} + \bar{x}^{\mathbf{b}} - t_i$ and carrying out algebraic manipulations, (2.4) can be rewritten as

$$(t_i - \bar{x}^{\mathbf{b}})^\top (x - \bar{x}^{\mathbf{b}}) \leq 0,$$

which is precisely the type of inequality generated by our exact cut generation algorithm, Algorithm 2.2. As discussed in the previous section, because the exact cut generation algorithm is not guaranteed to converge, we instead use inequalities

$$(t_i - \bar{x}^{\bar{\jmath}})^\top (x - \bar{x}^{\bar{\jmath}}) \leq b,$$

where $\|\bar{x}^{\bar{\jmath}} - \bar{x}^{\mathbf{b}}\| \leq \delta$. With algebraic manipulations, we rewrite this as

$$\|t_i - \bar{x}^{\bar{\jmath}}\| + \frac{(t_i - \bar{x}^{\bar{\jmath}})^\top}{\|t_i - \bar{x}^{\bar{\jmath}}\|}(x - t_i) \leq b,$$

showing that the inequalities generated by Algorithm 2.4 can be written as

$$G(t_i) + L(t_i)^\top (x - t_i) \leq b + \|t_i - \bar{x}^{\mathbf{b}}\| - \|t_i - \bar{x}^{\bar{\jmath}}\|,$$

where $L(t_i) := \frac{(t_i - \bar{x}^{\bar{\jmath}})^\top}{\|t_i - \bar{x}^{\bar{\jmath}}\|}$ is a function of t_i. We now show that the sequence $\{t_i\}_{i=1,\ldots,\infty}$ converges to a point x^c such that $G(x^c) \leq \epsilon$. Suppose not. At any iteration r, by construction the point t_k must satisfy all previous inequalities:

$$G(t_i) + L(t_i)^\top (t_r - t_i) \leq b + \|t_i - \bar{x}^{\mathbf{b}}\| - \|t_i - \bar{x}^{\bar{\jmath}}\|, \qquad \text{for all } 1 \leq i < r.$$

Then, there must exist some $\epsilon' > 0$, independent of r, such that

$$\epsilon + \epsilon' \leq G(t_i) \leq b + \|t_i - \bar{x}^{\mathbf{b}}\| - \|t_i - \bar{x}^{\bar{j}}\| + L(t_i)^\top (t_i - t_r) \leq b + \delta + L(t_i)^\top (t_i - t_r) < \epsilon + \delta + L(t_i)^\top (t_i - t_r),$$

where we used the fact that $b < \sqrt{\epsilon}^2$ if Algorithm 2.4 is executed using tolerance $\sqrt{\epsilon}$. Assume that in Algorithm 2.1 we reduce ζ (and hence, as a by-product, δ) every fixed number of iterations of Algorithm 2.4 at the same node.[1] If we are generating many cuts at a node without converging to a point x^c such that $G(x^c) \leq \epsilon$, the iteration index r grows until we have $\delta < \epsilon'$. Therefore, the above equation shows

$$\epsilon' - \delta < L(t_i)^\top (t_i - t_r) \leq \|L(t_i)\| \|t_i - t_r\| = \|t_i - t_r\|,$$

because $\|L(t_i)\| = 1$ by definition of L. In the above equation, for large r we have that $\epsilon' - \delta$ is strictly positive because δ gets progressively reduced. This implies that $\{t_i\}_{i=1,\ldots,\infty}$ does not contain a Cauchy subsequence, which is impossible because X is compact. It follows that $\{t_i\}_{i=1,\ldots,\infty}$ converges to a point x^c such that $G(x^c) \leq \epsilon$, i.e., $x^c \in \text{Conv}(C_x(\omega^k)) + \epsilon \cap B$, as desired.

The above discussion shows that for fixed ϵ, the number of calls to the cutting plane routine is finite before $\hat{x} \in \text{Conv}(C_x(\omega^k) \cap B) + \epsilon$ for every $k : z_k = 0$. We now need to show that the number of nodes in the branch-and-bound tree before the algorithm returns "infeasible", or a solution $(\hat{x^*}, \hat{z^*})$, is also finite. Note that whenever a candidate solution, say $(\hat{x}, \hat{z})$, is found such that $\hat{x} \in \text{Conv}(C_x(\omega^k) \cap B) + \epsilon$ for every selected scenario (i.e., such that $z_k = 0$), ϵ-feasibility for $\hat{x}$ is checked for all such scenarios. Hence, only ϵ-feasible solutions are accepted as incumbents. We now need to prove that the branching process is finite.

If $\hat{x}$ is ϵ-feasible, it is accepted and because we are solving a relaxation, it is optimal for the corresponding node. Suppose now that $\hat{x}$ is ϵ-infeasible; then Algorithm 2.1 branches on some component of $\hat{x}$. By assumption, we do not allow branching on components of $\hat{x}$ that are at one of their bounds; thus, even if no cutting occurs, after at most $O(n_x)$ branches (along each branching path) the point $\hat{x}$ will be a corner point of the hyperrectangle defined by the branching decisions B. When this happens, there are only two possibilities:

- if $\hat{x} \in \text{Conv}(C_x(\omega^k) \cap B) + \epsilon$ for all the selected scenarios, then $\hat{x}$ is an ϵ-feasible solution for these scenarios. Indeed, otherwise $\hat{x}$ would be the combination of at least two points in $(C_x(\omega^k) \cap B) + \epsilon$, which contradicts the fact that $\hat{x}$ is extreme for B;
- if $\hat{x} \notin \text{Conv}(C_x(\omega^k) \cap B) + \epsilon$ for some scenario k that has $z_k = 0$, we are able to generate a separating inequality (by assumption).

This shows that there is only a finite number of branching steps that can occur between the generation of cutting planes. We still need to show that we cannot switch between cutting and branching an infinite number of times; for this, we show that if we branch at some node N_1, when a cutting plane is generated at a child node N_2, we made "sufficient progress" with respect to N_1. Suppose that after solving the relaxation of node N_1, with branching decisions B_1, and obtaining the candidate solution $\hat{x}$, $\mathcal{C}$ does not return a cut and we are forced to branch. Because $\mathcal{C}$ fails to return a cut, $\hat{x} \in \text{Conv}(C_x(\omega^k) \cap B_1) + \epsilon$; let N_2 be a child node of N_1 where the new cutting plane is generated, with branching decisions B_2. Since a cut is generated for $\text{Conv}(C_x(\omega^k) \cap B_2) + \epsilon$, we must have $\text{Conv}(C_x(\omega^k) \cap B_2) + \epsilon \neq \text{Conv}(C_x(\omega^k) \cap B_1) + \epsilon$, and hence $C_x(\omega^k) \cap B_2 \neq C_x(\omega^k) \cap B_1$. Since

[1] For clarity of the presentation, this instruction is not explicitly given in Algorithm 2.1 but it is straightforward to include it.

$B_1 \subset B_2$, there exists some $x^h \in C_x(\omega^k) \cap B_1$ such that $x^h \notin C_x(\omega^k) \cap B_2$. In addition, we have that $\|\hat{x} - x^h\| > \epsilon$, because otherwise $\hat{x}$ would be ϵ-feasible: this implies that, by re-convexifying after branching, we have eliminated at least a ball of diameter ϵ, i.e., the feasible region at node N_2 has shrunk by at least a ball of diameter ϵ with respect to node N_1. Since all sets are compact, this shrinking process can only occur a finite number of times (upper bounded by the number of ϵ-balls necessary to cover the set). Thus, along each branching path, our algorithm can switch between branching and cutting only a finite number of times. This, together with standard arguments regarding the correctness of branch-and-bound, concludes the proof. ∎

Notice that the Decomposition Algorithm 2.1 checks ϵ-feasibility only. For ill-defined problems, it is possible that (2.3) is feasible while (CCP) is not; in this case, the algorithm terminates in a finite number of nodes, but it may return a solution declared to be optimal even if the (CCP) did not admit any such solution.

3. Algorithmic details and enhancements. In this section we complete the description of Algorithm 2.1 providing some additional details on the method, in particular concerning the branching strategy, cutting planes acceleration, node selection and heuristics. The computational effectiveness of all these elements will be evaluated in Section 4.

3.1. Branching strategy. Our branching strategy is based on the observation that the problem reduces to a deterministic one (i.e., without the chance constraint) whenever all z variables attain an integer value. For this reason, we first perform a standard branching on these variables, if some of them are not integer. Otherwise, we execute the separation procedure for the current point $\hat{x}$ and resort to spatial branching on an x_j variable only in Case 3, i.e., there is at least one active scenario for which $\hat{x}$ is infeasible though $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$. In this case, let ℓ_j and u_j denote the lower and upper bounds, respectively, for each variable x_j in the current subproblem. For branching, we consider as candidates all those variables such that $\ell_j < \hat{x}_j < u_j$. For each candidate variable and for each violated active scenario, we execute the separation procedure for the two nodes that would be generated by branching on that variable. All violated inequalities associated with a candidate variable are stored in a pool, and possibly added to the corresponding node subproblems whenever that variable is used for branching. The procedure selects for branching the first variable (if any) that would allow separation of $\hat{x}$ in both the generated nodes. If such a variable does not exist, the procedure arbitrary selects a branching variable that allows separation of $\hat{x}$ in at least one of the two descendant nodes. If such a variable does not exist either, the procedure selects an arbitrary branching variable among the candidates.

3.2. Cut generation. Recall that cut generation is possibly executed only in case all the z variables are integer. In this case, we check whether a master solution $\hat{x}$ can be separated from the continuous relaxation of some active scenario (Case 1), or from its convex hull (Case 2). Notice that explicitly handling the former case (Section 2.1) is not necessary for the correctness of the decomposition algorithm. Nevertheless, we include the generation of the corresponding inequalities because their separation typically requires a negligible effort compared with separation from the convex hull.

It is worth mentioning that any cut that is generated at a node of the branching tree whose feasible region has not been affected by spatial branching is globally valid. Conversely, in case some branching on x variables has been imposed, the resulting cut is only locally valid.

Next, we describe some ideas to speed-up the cut generation procedure.

3.3. Separation ordering. As the cut generation process may be time consuming, we halt it as soon as a valid inequality separating $\hat{x}$ is found for some active scenario. Thus, the order in which the active scenarios are processed may affect the overall performance of Algorithm 2.1.

To define an order that reflects the relevance of each scenario, in a preprocessing phase we compute the optimal solution value (cost) of the problem in which only one scenario is present at a time. The higher the associated value, the higher the priority of the scenario. In addition, preliminary experiments showed that when a specific scenario is not satisfied by the current solution, a number of inequalities are needed for producing a solution $\hat{x}$ that is feasible for that scenario. For this reason, at each node, the first active scenario that is considered for separation is the last violated scenario; the remaining scenarios are instead evaluated in the order given by the priorities.

3.4. Storing feasible points. At each execution, Algorithm 2.1 produces a collection of feasible points, whose convex combination includes $\hat{x}$ or defines a point minimizing the distance from $\hat{x}$. The procedure can be sped up by storing, for each scenario, a list of feasible points computed at any iteration. In addition, we use these points to check the feasibility of $\hat{x}$ for the active scenarios: if, for some scenario, the associated list contains a feasible point whose distance from $\hat{x}$ is smaller than ϵ, then $\hat{x}$ is feasible for that scenario; this approach avoids solving an optimization problem to check feasibility. In our implementation, we store up-to 10,000 points per scenario.

3.5. Node selection. Our node selection strategy gives priority to nodes that have been obtained from the root node by branching on the z variables only, i.e., without spatial branching conditions. Indeed, while cuts generated at these node can be globally added to the master problem, all inequalities generated after a spatial branching are only locally valid. In case the branching tree contains only nodes generated by spatial branching conditions, we select a node among those for which the separation procedure simulated at the parent node (see Section 3.1) produced some local cut. Otherwise, we let the solver select the node according to its internal strategy.

4. Application and Computational experiments. In this section, we evaluate the computational performances of our base algorithm and of the computational enhancements described in Section 3 on two problems derived from the literature. Specifically, we extend the linear problem addressed in [19], in which all variables are continuous, to define two new problems with binary and general integer variables, respectively. In both cases the objective function and the constraints are linear. We remark that a generalization to the nonlinear convex case is possible, as our analysis directly applies to this setting. However, this would require the use of nonlinear convex MINLP solvers, that are typically less stable than MILP solvers from a numerical point of view, and would make the computational analysis less reliable; thus, we test the linear case only. The scope of these experiments is twofold: first, we are interested in assessing the effectiveness of the proposed computational enhancements, and to understand what is the limit of the methodology we propose. Second, we want to compare the performance of the proposed methodology with an alternative solution approach consisting in the direct application of a general-purpose MILP solver to the formulation (CCP-MINLP).

All algorithms are implemented in C++ and use the `CPLEX 12.7.1` framework to implement our branch-and-cut algorithm. The experiments are performed on a computer equipped with an i7 processor clocked at 3.20 GHz and 64 GB RAM running Linux. The master problem and each scenario subproblem are solved in single-threaded mode, imposing, for the master problem, a tree-size limit to 16 GB and disabling its heuristic procedures; heuristics are disabled because in a preliminary computational evaluation they were found ineffective, leading to an overall slowdown of the proof of optimality. All the remaining CPLEX parameters are left to their default values.

4.1. Application: Resource Planning. In [19], the author considers a continuous resource planning problem inspired by a call center staffing application. The input consists of a set of resources used to meet demands for a set of customer types, and the objective is to determine the quantity of each resource to activate to satisfy the demands. We consider two variants of this application that have binary and integer variables. Both problems consider a set $I = \{1, \dots, n\}$ of resources and a set $J := \{1, \dots, m\}$ of customer types. Given the unit cost c_i of each resource i, the demand λ_j of each customer type j, and the service rate μ_{ij} of resource i for customer type j, the first problem is to decide the activation level of each resource and the assignment of customers to resources, in such a way that the demand of the customers is satisfied at minimum cost. In the first problem (denoted as INT), we introduce variables $x_i,\ i \in I$, and $y_{ij},\ i \in I,\ j \in J$ to define the activation level of resource i and the amount of resource i allocated to customer type j, respectively. The y variables are imposed to be integer, i.e., resources must be assigned to customers in integer multiples, producing the following formulation:

$$
\begin{aligned}
\min \quad & \sum_{i \in I} c_i x_i & & & & \\
\text{s.t.:} \quad & \sum_{i \in I} \mu_{ij} y_{ij} & \geq & \lambda_j & & j = 1 \dots m \\
& \sum_{j \in J} y_{ij} & \leq & x_i & & i = 1 \dots n \\
& y_{ij} & \in & \mathbb{N} & i = 1 \dots n, \quad & j = 1 \dots m
\end{aligned}
\tag{INT}
$$

In the second problem (denoted as BIN), the y variables are imposed to be binary, so that each customer type is assigned exactly one resource. The continuous variables $x_i,\ i \in I$ denote the activation level of each resource i as in the first problem. For each customer type j and resource i, the amount of resource used by the customer type if the assignment is performed is given by $\left\lceil \frac{\lambda_j}{\mu_{ij}} \right\rceil$. The formulation reads as

$$
\begin{aligned}
\min \quad & \sum_{i \in I} c_i x_i & & & & \\
\text{s.t.:} \quad & \sum_{i \in I} y_{ij} & = & 1 & & j = 1 \dots m \\
& \sum_{j \in J : \mu_{ij} > 0} \left\lceil \frac{\lambda_j}{\mu_{ij}} \right\rceil y_{ij} & \leq & x_i & & i = 1 \dots n \\
& y_{ij} & \in & \{0, 1\} & i = 1 \dots n, \quad & j = 1 \dots m
\end{aligned}
\tag{BIN}
$$

In the stochastic version of the two problems, we assume the demand of each customer type to be a random parameter with discrete probability distribution over a set $K = \{1, \dots, h\}$ of scenarios. Hence, we denote by λ_{jk} the demand of customer type j in case scenario $k \in K$ materializes. The problems are approached in two stages; in the first one, the activation of the resources is decided, before the actual demands are revealed, whereas in the second stage customer types are assigned to activated resources. For each problem, the first set of constraints is imposed as a set of chance constraints, and feasibility of each scenario $k \in K$ is achieved if it is possible to assign all the demands (in the first problem) or all the customer types to the resources activated in the first stage (in the second problem).

4.2. Test instances. We define our benchmark for problems INT and BIN starting from the original instances introduced in [19] together with a large set of distinct scenarios. Given an original instance and a number h of scenarios, we use the cost coefficients c_i from the original data, whereas demands are derived by considering the first h scenarios, dividing each entry by 10 and rounding down the resulting values. Accordingly, for the service rates, we replace each non-zero entry in the base instance with a random value between 1 and 10.

We evaluate each of the five original instances with $n = 20$ resources and $m = 30$ customer types with different values for the number h of scenarios ($h \in \{10, 20, 50, 100, 200\}$ for INT and $h \in \{10, 20, 50, 100\}$ for BIN) and for the risk level ($\alpha \in \{0.05, 0.10, 0.20\}$). Observe that there are no instances with $h = 10$ and $\alpha = 0.05$ as this would correspond to the deterministic case in which no scenario can be violated. Thus, our benchmark includes 70 instances for problem INT and 55 instances for problem BIN.

4.3. Problem-specific primal heuristics. We design a problem-specific primal heuristic, referred to as `Fast`, to compute an initial feasible solution. The heuristic, applied at the root, works as follows. Initially, it solves each scenario, one at a time, and sorts the scenarios according to non-increasing values of their optimal solution. Then, it processes the scenarios in the same order, and solves each of them while, iteratively, the lower bound of each first stage variable x_j is set to the optimal value attained by that variable in the solution of the previous scenario. Since in the application we consider, the x variables represent resources that do not have an upper bound, the `Fast` heuristic defines a feasible solution.

A similar procedure is used at all nodes of the branch-and-bound tree before a spatial branching takes place. The only difference is that the (infeasible) solution $\hat{x}$ of that node is used to initialize the lower bounds for the x variables when the first scenario is solved.

4.4. Algorithm tuning. In the first set of experiments, we investigate the effectiveness of the components embedded in our branch-and-cut algorithm that we will denote as `DEC`. The algorithm includes all the enhancements described in Section 3, and the problem-specific heuristic `Fast`, applied both at the root and at subsequent nodes.

In the following, we deactivate each component of algorithm `DEC`, one at a time, in order to assess its contribution to the algorithm performance. In particular, we consider the following variants:

- `FrOrd` is the algorithm obtained removing the scenario ordering strategy, and ordering the scenarios as in [19], i.e., by decreasing values of the total number of times that each scenario has yielded a violated inequality (the higher the frequency, the better);
- `NoHeur` is obtained disabling the execution of the problem-specific primal heuristics;
- `NoPoint` corresponds to the version in which feasible points are not stored for later use;
- `CpxNode` is obtained by removing the node selection strategy that favors nodes not affected by spatial branching (and letting the MIP solver decide on node selection);
- `RandSB` replaces the spatial branching selection strategy with a random selection of the variable used for spatial branching;
- `Basic` is the version of the algorithm in which all the features above are deactivated at the same time.

We execute each variant of the algorithm on the instances of problems INT and BIN with 50 scenarios, with a time limit of 10 hours for each run. Table 1 reports, for each problem and set of 5 instances with the same value of α, the number of proven optimal solutions and the average computing time (over instances solved to optimality) of each algorithm tested.

TABLE 1
Comparison of the performances of different variants of the algorithm on instances with 50 scenarios.

		DEC		FrOrd		NoHeur		NoPoint		CpxNode		RandSB		Basic	
	α	Opt	Time	Opt	Time	Opt	Time	Opt	Time	Opt	Time	Opt	Time	Opt	Time
INT	0.05	5	313.0	5	824.7	5	403.6	5	1221.0	5	312.2	4	66.2	4	1863.8
	0.10	5	201.8	5	698.5	5	659.9	5	649.7	5	846.6	4	470.3	2	8175.4
	0.20	5	599.8	5	1592.1	5	1524.5	5	2474.3	5	2358.5	2	374.5	3	22296.2
BIN	0.05	4	3318.6	4	2630.2	4	4884.6	4	4837.9	4	3921.3	-	-	-	-
	0.10	3	2672.9	3	5462.2	2	3951.5	2	2847.2	3	4529.8	-	-	-	-
	0.20	4	14491.7	3	11364.7	2	14853.5	3	15471.8	2	6655.2	-	-	-	-

The results show that algorithm DEC is able to solve all instances for problem INT in less than 600 seconds, on average. For problem BIN, algorithm DEC solves to optimality three instances for $\alpha = 0.10$, and four instances for $\alpha = 0.05$ and for $\alpha = 0.20$; in this last case, the average computing time increases to 4 hours. Algorithm FrOrd shows that disabling the scenario order strategy produces a considerable worsening of the performances: this version solves to optimality one fewer instance and has more than double the average computing time for instances of problem INT and for instances of problem BIN with $\alpha = 0.10$; it is faster than DEC only for instances of problem BIN with $\alpha = 0.05$, by a few hundred seconds. Similarly, we notice a considerable worsening of the performances when removing primal heuristics: algorithm NoHeur has an average computing time for problem INT that is more than double the one of DEC, for $\alpha \in \{0.10, 0.20\}$. In addition, it solves to optimality fewer instances of problem BIN (one fewer instance for $\alpha = 0.10$ and two fewer instances for $\alpha = 0.20$). Algorithms NoPoint and CpxNode exhibit a similar behavior: the average computing time for instances of problem INT is increased between 2 and 4 times compared to DEC, while for problem BIN, the number of instances solved to optimality is smaller. The central role of our refined strategy in spatial branching is clear comparing algorithms DEC and RandSB. Indeed, the latter solves to optimality only 10 instances (out of 15) for problem INT, and no instance for problem BIN. The last version, obtained disabling all the features of DEC, also shows poor performance: algorithm Basic is able to solve to optimality only 9 instances of problem INT, and no instance of problem BIN. Summing up, the results reported in Table 1 clearly show that each component of algorithm DEC gives a valuable contribution to the computational performance of the method, the most critical being the spatial branching strategy. In addition, the table shows that a naive implementation of Algorithm 2.1, similar to our Basic variant, is far from being effective on these problems.

4.5. Performance of spatial branching. As already mentioned, a careful spatial branching strategy contributes significantly to the effectiveness of our algorithm DEC. Here, we present some additional results aimed at evaluating the relevance of spatial branching in more detail, by comparing with a version of the algorithm in which spatial branching is not implemented at all. In this scheme, denoted as NoSB in the following, a node of the branching tree is fathomed if the solution of the continuous relaxation of the node is infeasible after the separation on the continuous relaxation and on the convex hull. In this case, the incumbent is updated, i.e., this scheme returns the value of the lower bound, say L, obtained optimizing over the intersection of the convex hulls of the selected scenarios. Comparing this lower bound with the optimal solution value provides a

measure of the gap that has to be closed by spatial branching.

Table 2 reports the results for algorithms NoSB and DEC on the instances of our benchmark. Each row of the table addresses the 5 instances of the same problem defined by the same values for h and α. For both algorithms, columns "Cont" and "Conv" report the number of cuts generated on the continuous relaxation and on the convex hull, respectively. In addition, for NoSB we report the corresponding gap percentage, computed as gap $= 100 \times (z^* - L)/L$, where z^* is the optimal solution value. For algorithm DEC, we also report the number of times that spatial branching is performed ("# bra") and the total number of branch-and-bound nodes explored ("# nodes"). All entries report averages (rounded to the closest integer) computed over all the instances solved to optimality by algorithm DEC.

TABLE 2
Computational details of algorithm DEC.

			NoSB			DEC			
	h	α	Cont	Conv	gap	Cont	Conv	# bra	# nodes
INT	10	0.10	370	78	0.01	354	94	2	8
		0.20	516	55	0.00	399	79	1	11
	20	0.05	511	82	0.00	336	64	1	4
		0.10	439	84	0.00	442	106	8	23
		0.20	711	106	0.03	562	108	11	55
	50	0.05	630	99	0.02	520	136	9	26
		0.10	821	121	0.02	521	160	7	106
		0.20	1439	197	0.03	782	298	21	342
	100	0.05	1092	168	0.06	549	420	42	136
		0.10	1351	219	0.02	681	262	18	492
		0.20	2262	296	0.02	1523	457	32	1484
	200	0.05	1595	240	0.02	751	283	12	423
		0.10	2361	331	0.03	774	454	35	3057
		0.20	3507	531	0.01	2271	843	69	19863
BIN	10	0.10	79	0	0.12	188	269	106	205
		0.20	69	0	0.33	324	856	162	308
	20	0.05	76	0	0.31	364	696	129	235
		0.10	78	0	0.44	649	1125	283	534
		0.20	121	0	0.32	503	748	222	421
	50	0.05	109	0	0.44	645	1270	266	470
		0.10	133	0	0.30	722	1222	256	509
		0.20	201	0	0.45	1979	4657	1120	2271
	100	0.05	136	0	0.69	1327	5228	709	1286
		0.10	211	0	0.52	1573	4343	1158	2453
		0.20	287	0	0.47	2345	4554	1219	2822

The results confirm that problem INT is easier to solve than BIN. Indeed, event without spatial

branching, a few thousand cuts in total are sufficient to obtain very small gaps. Algorithm `DEC` rarely has to resort to spatial branching: it takes place, on average, only 69 times on the most difficult instances ($h = 200$ and $\alpha = 0.20$). For this group of instances, the average number of generated branch-and-bound nodes is below 20,000.

Concerning problem BIN, without spatial branching we still attain fairly small residual percentage gaps, the largest average gap being 0.69 for the instances with $h = 100$ and $\alpha = 0.05$. Observe that the number of cuts that are generated on the continuous relaxation is relatively small, and that no cuts are generated on the convex hull. Indeed, the continuous relaxation of problem BIN always admits an optimal solution that is integral, corresponding to the assignment of each customer type j to the resource i for which $c_i \left\lceil \frac{\lambda_j}{\mu_{ij}} \right\rceil$ is a minimum. Hence, extreme points of the continuous relaxation are integer, and the continuous relaxation coincides with the convex hull of the integer solutions; this is the case at the root node, before any spatial branching takes place. When active, spatial branching changes the bounds of the x variables; as a consequence, the continuous relaxation of the problem does not necessarily admit optimal integer solutions after branching. In this setting, few hundred spatial branchings are performed for instances with $h \leq 20$, together with the generation of few hundred cuts on the continuous relaxation and on the convex hull. The number of necessary cuts and spatial branchings increases considerably (up to one order of magnitude) when considering larger instances with $h \geq 50$.

4.6. Computational comparison. In this section, we study the possibility to further enhance the performance of algorithm `DEC`. To this end, we implement a hybrid strategy, denoted `DEC+`, that combines `DEC` with the use of a MIP solver for computing an initial feasible solution at the root node and for solving deterministic subproblems at the subsequent nodes.

Algorithm `DEC+` executes a heuristic procedure (that we call `Complete`) at the root node. This procedure: (i) orders the scenarios with the same rule used by the heuristic `Fast` (see Section 4.3); (ii) defines a deterministic problem by selecting the scenarios (i.e., fixing the associated z variables) in the given order, until the chance constraint is satisfied; and (iii) solves this deterministic problem using a MIP solver as a black box. Notice that procedure `Complete` can be applied to any problem instance, but it can be quite slow in practice, as the deterministic problem must enforce feasibility for all the selected scenarios and may thus include a very large number of variables and constraints. In addition, algorithm `DEC+` fully solves (defining a sub-MIP) branch-and-bound nodes in which all the z variables are fixed because of branching. Indeed, each such node induces a deterministic problem, and we can immediately close the node if the deterministic problem is solved, after possibly updating the incumbent.

We compare the performance of `DEC` and `DEC+` with the direct application of a MIP solver (the same solver that is used by our algorithm, i.e., IBM-ILOG CPLEX 12.7.1) on formulation (CCP-MINLP). The solver, denoted as `CPLEX` in the reminder of this section, is executed in single-thread mode with all the remaining parameters to their default values.

Table 3 reports the outcome of our experiments on all instances introduced in Section 4.2 for problems INT and BIN. As in previous experiments, we use a time limit of 10 hours per run. Each row of the table refers to 5 instances of the same problem, having the same number h of scenarios and the same risk parameter α. For each solution method, the table reports the number of optimal solutions, the average computing time (with respect to instances solved to optimality only) and the average percentage optimality gap, computed as gap $= 100 \times (U - L)/L$. In 5 of the 15 instances of problem BIN with $h = 100$, `DEC+` reaches the time limit during the execution of the heuristic procedure `Complete` at the root node. Because the method is unable to compute a lower bound in

these cases, we calculated the average optimality gap with respect to the remaining instances only (and put the number of instances used to calculate the gap in brackets).

TABLE 3
Comparison of the performances of different solution approaches.

			DEC			DEC+				CPLEX		
	h	α	Opt	Avg time	Avg gap	Opt	Avg time	Avg gap		Opt	Avg time	Avg gap
INT	10	0.10	5	77.41	0.00	5	32.57	0.00		5	5.63	0.00
		0.20	5	66.76	0.00	5	20.19	0.00		5	11.80	0.00
	20	0.05	5	53.52	0.00	5	28.13	0.00		5	12.13	0.00
		0.10	5	236.43	0.00	5	61.79	0.00		5	36.92	0.00
		0.20	5	208.83	0.00	5	99.95	0.00		5	35.69	0.00
	50	0.05	5	313.01	0.00	5	277.58	0.00		5	147.98	0.00
		0.10	5	201.85	0.00	5	108.55	0.00		5	263.59	0.00
		0.20	5	599.84	0.00	5	389.96	0.00		4	577.95	0.00
	100	0.05	5	1715.09	0.00	5	245.42	0.00		5	1248.72	0.00
		0.10	5	549.45	0.00	5	281.80	0.00		5	4225.70	0.00
		0.20	5	1378.61	0.00	5	646.13	0.00		5	5504.79	0.00
	200	0.05	5	1010.50	0.00	5	1358.12	0.00		4	9775.20	0.00
		0.10	5	1321.17	0.00	5	3016.71	0.00		3	31361.04	3.97
		0.20	5	3513.01	0.00	5	6066.73	0.00		-	-	10.39
BIN	10	0.10	5	473.26	0.00	5	29.20	0.00		5	11.30	0.00
		0.20	3	815.82	0.40	5	12.00	0.00		5	24.89	0.00
	20	0.05	4	814.93	0.06	5	260.11	0.00		5	145.04	0.00
		0.10	5	1763.06	0.00	5	424.08	0.00		5	434.94	0.00
		0.20	5	1697.42	0.00	5	198.52	0.00		5	387.29	0.00
	50	0.05	4	3318.60	0.08	5	961.91	0.00		5	4654.40	0.00
		0.10	3	2672.95	0.21	5	5758.61	0.00		5	14716.14	0.00
		0.20	4	14491.71	0.24	5	1367.80	0.00		4	11796.62	0.33
	100	0.05	1	16084.49	1.44	1	12507.32	6.13	(3)	-	-	3.52
		0.10	1	20089.03	0.89	2	4494.95	8.51	(4)	-	-	4.24
		0.20	2	22010.25	0.95	1	3728.20	6.61	(3)	-	-	7.80

Both DEC and DEC+ are able to solve all the instances of problem INT. However, DEC+ is typically faster for instances with a small number of scenarios, as in this case the deterministic subproblems can be solved effectively, while it turns out to be slower for instances with $h = 200$. Not surprisingly, CPLEX is the fastest method for small instances ($h \in \{10, 20\}$) whose associated (CCP-MINLP) MIP formulation is manageable. Increasing the number h of scenarios produces larger number of variables in the model that make the direct use of the solver impractical. Indeed, CPLEX is not able to solve to optimality all the instances with $h = 50$, and no instance is solved for $h = 200$ and $\alpha = 0.20$.

Problem BIN is more challenging for all solution methods. Comparing DEC and DEC+, the latter

seems to have a better performance: it solves to optimality all the instances with $h \leq 50$, and it is much faster than DEC, which is unable to solve 7 of the 40 instances to optimality. Both variants of the algorithm can solve 4 instances with $h = 100$. However, for this latter group of instances, algorithm DEC is always able of computing a feasible solution, with corresponding optimality gaps ranging from 0.95 to 1.44. Instead, there are 5 instances for which algorithm DEC+ hits the time limit during the search for an initial solution, and no feasible solution is available. Concerning CPLEX, its performance is comparable with that of DEC+ for instances with $h \leq 20$. Then, it solves all but one instance with $h = 50$, but it is one order of magnitude slower than DEC+. Results get worse for $h = 100$, as no instance is solved in this case: the corresponding optimality gap is in the range from 3.52 to 7.80, confirming that the direct application of the solver to formulation CCP-MINLP leads to gaps that can be unsatisfactory when the number of scenarios increases.

One may wonder if the solution of nodes with fixed z variables is the main workhorse of the DEC+ algorithm, or in other words, if the cutting planes valid for the convex hull of the scenario subproblems are still beneficial in the setting of DEC+. We experimentally verified that, on problem INT, where these cuts are used starting from the root node, disabling the cut generation procedure on the convex hull leads to a significant worsening of the algorithm's performance. Indeed, in this setting, the resulting algorithm is able to solve, within the time limit, only 5 instances with $h = 50$ and no instance with $h = 100$, witnessing that the cuts are very impactful even in the setting of DEC+.

5. Conclusions and future work. In this work, we considered chance constrained mathematical problems in which the second-stage variables are integer and uncertainty is modeled via a discrete set of scenarios. The feasible regions of the second-stage problems are not convex sets, as they are defined by convex constraints and by the integrality of the second-stage variables. To the best of our knowledge, this is the first attempt to solve this class of problems with general integer second-stage variables, which are known to be difficult in practice and tend to complicate the solution algorithm. We introduced a decomposition approach based on a branch-and-cut framework. In this framework, we possibly generate cutting planes based on outer approximation of the convex hull of the integer points, and resort to spatial branching when the current infeasible solution cannot be separated using a linear cut. We analyzed the theoretical convergence of the algorithm both in case of infinite precision and when some numerical tolerances are used. Finally, we performed an extensive computational analysis on two problems derived from a test-case addressed in the recent literature, showing that our algorithm is able to solve medium-sized instances in a reasonable amount of time, and much more effectively than a state-of-the-art solver applied to the deterministic equivalent of the chance-constrained problem. To achieve this performance we used many computational enhancements, all of which were empirically shown to be beneficial in improving the performance of the algorithm.

REFERENCES

[1] J. F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematik, 4 (1962), p. 238–252.

[2] P. Beraldi and A. Ruszczyński, *A branch and bound method for stochastic integer problems under probabilistic constraints*, Optimization Methods and Software, 17 (2002), pp. 359–382.

[3] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*, Springer Science & Business Media, 2011.

[4] C. C. Carøe and J. Tind, *L-shaped decomposition of two-stage stochastic programs with integer recourse*, Mathematical Programming, 83 (1998), pp. 451–464.

[5] A. Charnes and W. W. Cooper, *Deterministic equivalents for optimizing and satisficing under chance constraints*, Operations research, 11 (1963), pp. 18–39.
[6] D. Dentcheva, A. Prékopa, and A. Ruszczynski, *Concavity and efficient points of discrete distributions in probabilistic programming*, Mathematical Programming, 89 (2000), pp. 55–77.
[7] M. Duran and I. Grossmann, *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming, 36 (1986), pp. 307–339.
[8] D. Gade, S. Küçükyavuz, and S. Sen, *Decomposition algorithms with parametric gomory cuts for two-stage stochastic integer programs*, Mathematical Programming, 144 (2014), pp. 39–64.
[9] A. M. Geoffrion, *Generalized benders decomposition*, Journal of optimization theory and applications, 10 (1972), pp. 237–260.
[10] R. G. Jeroslow, *Representability in mixed integer programming, I: characterization results*, Discrete Applied Mathematics, 17 (1987), pp. 223–243.
[11] J. E. Kelley, *The cutting-plane method for solving convex programs*, Journal of the Society of Industrial and Applied Mathematics, 8 (1960), pp. 703–712.
[12] F. Kilinç-Karzan, S. Küçükyavuz, and D. Lee, *Decomposition algorithms with parametric gomory cuts for two-stage stochastic integer programs*, Tech. Rep, (2021).
[13] S. Küçükyavuz, *On mixing sets arising in chance-constrained programming*, Mathematical programming, 132 (2012), pp. 31–56.
[14] S. Küçükyavuz and R. Jiang, *Chance-constrained optimization: a review of mixed-integer conic formulations and applications*, Tech. Rep, (2021).
[15] S. Küçükyavuz and S. Sen, *An introduction to two-stage stochastic mixed-integer programming*, in Leading Developments from INFORMS Communities, INFORMS, 2017, pp. 1–27.
[16] G. Laporte and F. V. Louveaux, *The integer l-shaped method for stochastic integer programs with complete recourse*, Operations Research Letters, 13 (1993), pp. 133 – 142.
[17] X. Liu, S. Küçükyavuz, and J. Luedtke, *Decomposition algorithms for two-stage chance-constrained programs*, Mathematical Programming, 157 (2016), pp. 219–243.
[18] A. Lodi, E. Malaguti, G. Nannicini, and D. Thomopulos, *Nonlinear chance-constrained problems with applications to hydro scheduling*, Mathematical Programming, (2019), https://doi.org/10.1007/s10107-019-01447-3.
[19] J. Luedtke, *A branch-and-cut decomposition algorithm for solving chance-constrained mathematical programs with finite support*, Mathematical Programming, 146 (2014), pp. 219–244.
[20] J. Luedtke and S. Ahmed, *A sample approximation approach for optimization with probabilistic constraints*, SIAM Journal on Optimization, 19 (2008), pp. 674–699.
[21] J. Luedtke, S. Ahmed, and G. L. Nemhauser, *An integer programming approach for linear programs with probabilistic constraints*, Mathematical programming, 122 (2010), pp. 247–272.
[22] V. I. Norkin, Y. M. Ermoliev, and A. Ruszczyński, *On optimal allocation of indivisibles under uncertainty*, Operations Research, 46 (1998), pp. 381–395.
[23] A. Prekopa, *Contributions to the theory of stochastic programming*, Mathematical Programming, 4 (1973), pp. 202–221.
[24] A. Prékopa, *Dual method for the solution of a one-stage stochastic programming problem with random rhs obeying a discrete probability distribution*, Zeitschrift für Operations Research, 34 (1990), pp. 441–461.
[25] Y. Qi and S. Sen, *The ancestral Benders' cutting plane algorithm with multi-term disjunctions for mixed-integer recourse decisions in stochastic programming*, Mathematical Programming, 161 (2017), p. 193–235.
[26] S. Sen, *Relaxations for probabilistically constrained programs with discrete random variables*, Operations Research Letters, 11 (1992), pp. 81–86.
[27] S. Sen and J. L. Higle, *The c 3 theorem and a d 2 algorithm for large scale stochastic mixed-integer programming: Set convexification*, Mathematical Programming, 104 (2005), pp. 1–20.
[28] Y. Song, J. R. Luedtke, and S. Küçükyavuz, *Chance-constrained binary packing problems*, INFORMS Journal on Computing, 26 (2014), pp. 735–747.
[29] R. M. Van Slyke and R. Wets, *L-shaped linear programs with applications to optimal control and stochastic programming*, SIAM Journal on Applied Mathematics, 17 (1969), pp. 638–663.
[30] M. Zhang and S. Küçükyavuz, *Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs*, SIAM Journal on Optimization, 24 (2014), pp. 1933–1951.