



An algorithm for binary linear chance-constrained problems using IIS

Gianpiero Canessa¹ · Julian A. Gallego² · Lewis Ntaimo³ · Bernardo K. Pagnoncelli^{4,5}

Received: 8 December 2017 / Published online: 2 January 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

We propose an algorithm based on infeasible irreducible subsystems to solve binary linear chance-constrained problems with random technology matrix. By leveraging on the problem structure we are able to generate good quality upper bounds to the optimal value early in the algorithm, and the discrete domain is used to guide us efficiently in the search of solutions. We apply our methodology to individual and joint binary linear chance-constrained problems, demonstrating the ability of our approach to solve those problems. Extensive numerical experiments show that, in some cases, the number of nodes explored by our algorithm is drastically reduced when compared to a commercial solver.

Keywords Chance-constrained programming · Infeasible irreducible subsystems · Integer programming

1 Introduction

Chance-constrained programming (CCP) is a modelling framework to address optimization problems under uncertainty. The resulting problems are notoriously difficult to solve, mainly due to the lack of convexity in the general case. Even for continuous

Pagnoncelli and Canessa thank the support of Anillo Grant ACT 1407 and PienSA Minería 2018 Grant.

✉ Gianpiero Canessa
gianpiero.canessa@edu.uai.cl

¹ Universidad Adolfo Ibáñez, Diagonal Las Torres 2640, 7941169 Santiago, Chile

² AT Kearney Inc, 227 West Monroe Street, Chicago, IL 60606, USA

³ Texas A&M University, 3131 TAMU, Office ETB 4008, College Station, TX 77843, USA

⁴ Universidad Adolfo Ibáñez, Diagonal Las Torres 2640 Oficina 533-C, 7941169 Santiago, Chile

⁵ IEMS Department, Northwestern University, 2145 Sheridan Road Tech C216, Evanston, IL 60208-3109, USA

problems, explicitly evaluating whether a candidate point is feasible is challenging, and Monte-Carlo methods are often employed. Aside from very special cases, it is challenging to solve CCP to optimality, and approximations need to be considered.

CCP was first proposed in [8] and has been extensively studied since then. We believe the main reason for its popularity is that CCP represents a very natural and intuitive way for modelling uncertainty. In the typical case, the decision maker optimizes a deterministic function, and randomness is present in the constraints, with known distribution. A given point is feasible whenever the constraints are satisfied with (at least) some pre-defined probability level. In other words, the search of the optimal solution is restricted to points that satisfy the random constraints for a “large” percentage of the realizations of the random vector.

Aside from being an intuitive modelling tool, CCP is simpler than two-stage stochastic programming problems in terms of parameter specification. In some situations there is no recourse action after a decision is made, and even when recourse is available, estimating the second stage cost coefficients can be challenging. Finally, CCP captures risk aversion since a chance constraint can be viewed as a Value-at-Risk (VaR) constraint.

A general linear CCP problem can be written as follows:

$$\begin{aligned} & \min f(x) \\ & \text{s.t. } Ax \geq b, \\ & \quad P\{T(\omega)x \geq h(\omega)\} \geq 1 - \alpha, \\ & \quad x \in \mathcal{X}. \end{aligned} \tag{1}$$

The objective function $f(\cdot)$ is convex in $x \in \mathbb{R}^n$, and in most applications it is linear. The constraints defined by matrix $A_{m_1 \times n}$ and vector $b_{m_1 \times 1}$ represent the m_1 deterministic constraints of the problem. For instance in a portfolio problem one could impose no short sales, and that the sum of all investments has to be equal to one. We assume ξ is a d -dimensional random vector with probability distribution P supported on a set $\mathcal{E} \subset \mathbb{R}^d$, T is a $m_2 \times n$ matrix and h is a $m_2 \times 1$ vector. When $m_2 > 1$ we have joint chance constraints, which will be dealt with separately in this paper since this case needs special treatment. The chance constraint has to be satisfied with probability at least $1 - \alpha$, where $\alpha \in [0, 1)$ is the desired reliability level defined by the decision maker. Finally, the set \mathcal{X} can be continuous, integer or binary.

Several algorithms have been proposed for different versions of the problem. When matrix T has dimension $1 \times n$ and ξ follows a multivariate normal distribution, the chance constraint can be converted into a second order conic constraint, which can be solved efficiently by off-the-shelf solvers [1,13,22]. For problems where randomness is only present on vector h , the concept of p -efficient points can be applied, which allows the construction of tractable equivalent formulations for the CCP (see [10] for the discrete case and [18] for the continuous one). The method was extended by [14] to include mixed-integer variables.

The Sample Average Approximation (SAA) is a popular approach whose advantages are that both matrix T and vector h can be random, and that the distribution is arbitrary, as long as samples can be obtained from it [16,17]. When the original distri-

bution is continuous, or discrete with a very large number of scenarios, SAA consists in generating samples and constructing an approximate problem that is tractable. It can be shown that under mild conditions the optimal solution (or set of solutions) and optimal value of the sampled problem converge to their deterministic counterparts as the sample size increases. CCP continues to attract the attention of researchers, and recent publications focused on other approaches such as the study of mixing sets [2,12] and boolean functions [15], among others.

The class of problems that received less attention are the ones with pure binary variables, that is, $\mathcal{X} = \{0, 1\}^n$. In [3,20] the authors propose algorithms for specific versions of important binary problems. In [19] the authors propose an algorithm based on p -inefficiency points to solve several instances of the probabilistic set covering problem. The focus of this paper is on a new general algorithm for linear CCP with binary decision variables. The CCP can have either separated or joint chance-constraints, and in both cases the technology matrix and right hand-side vector can be random. The main idea is to use infeasible irreducible subsystems (IIS) to obtain cuts that can speed the convergence of the algorithm. IIS were used in the context of CCP in [21], and the authors report significant reductions in terms of nodes explored and time with respect to the determinist equivalent formulation. However, the authors address the $\mathcal{X} = \mathbb{R}^n$ case with an individual chance constraint, and do not consider binary variables in the formulation.

Our methodology uses the functionalities of commercial solvers to generate IIS cuts. When the problem is infeasible with all scenario constraints included, the IIS identifies the source of infeasibility and indicates the constraints that should be removed to achieve feasibility. Interestingly, our methodology also addresses the feasible case, that is, even if a solution is available when all scenario constraints are considered, IIS can help find better solutions by removing some scenarios. For the feasible case, we introduce an additional constraint on the objective function value, and look for solutions that improve on the best upper bound obtained so far. In addition, our approach uses the fact that since $\mathcal{X} = \{0, 1\}^n$, we are able to estimate the minimal improvement that can be achieved in terms of the objective function value at each step. Through extensive computational experiments performed on separated and joint chance-constrained problems, we show that our approach can significantly decrease the number of nodes explored when compared to solving the problem directly using a commercial solver.

The rest of the paper is organized as follows. We start by defining IIS, providing examples and building the connection with CCP. In Sect. 3 we describe the formulations we will be working with, and the results we need in order to establish the validity of our approach. In Sect. 4 we describe the algorithm and discuss some implementation aspects. Section 5 presents the numerical experiments, and finally Sect. 6 concludes the paper.

2 Infeasible irreducible subsystems

In this section we give a brief introduction on IIS, and discuss how this idea can be applied to chance-constrained problems.

2.1 The deterministic equivalent formulation

In this work, we focus on chance-constrained problems with linear objective function and linear constraints. When the distribution of the random parameters is finite and belongs to set Ω , with $|\Omega| = S$, problem (1) can be written in the following deterministic equivalent formulation:

$$\begin{aligned}
 \text{DEP: } \min \quad & c'x \\
 \text{s.t. } \quad & Ax \geq b, \\
 & T(\omega)x + Mz_\omega \geq h(\omega), \quad \forall \omega \in \Omega, \\
 & \sum_{s \in \Omega} p_\omega z_\omega \leq \alpha, \\
 & z_\omega \in \{0, 1\}^S, \quad x \in \mathcal{X},
 \end{aligned} \tag{2}$$

where p_ω is the probability of each element in Ω , and M is a constant that guarantees feasibility whenever z_ω is equal to one. Problem (2) can also be used when the distribution of the random parameters is continuous, or discrete with an infinite number of possible realizations. In those cases, a sample of size S is generated and formulation (2) can be regarded as an instance of a SAA problem. It can be shown (e.g. [16]) that optimal solutions and the optimal value converge to their exact counterparts. Guidelines can be found in [6,7] as to how to choose the sample size S such that with arbitrarily high probability the solution obtained in the SAA instance is feasible to the original problem.

For simplicity, we refer to formulation (2) as the deterministic equivalent formulation (DEP). For moderate sizes of S , the DEP can be solved directly; it is often the case that such approach is not efficient, mostly because the formulation contains the so-called big- M constraint, and relaxations tend to be very poor. The focus of this work is to solve the DEP efficiently when \mathcal{X} is binary, and in what follows we will discuss how IIS can help on this task.

2.2 Irreducibly infeasible subsystems

Given a set of constraints Q , an IIS is a subset $P \subseteq Q$ such that

- (a) It is infeasible;
- (b) Any constraint p that is removed from P turns the resulting subsystem $P \setminus \{p\}$ feasible.

Example Consider the following set of constraints Q (Fig. 1):

$$Q = \{x_1 - x_2 \leq 0, \tag{A}$$

$$2x_2 \leq 1, \tag{B}$$

$$x_1 + x_2 \geq 2, \tag{C}$$

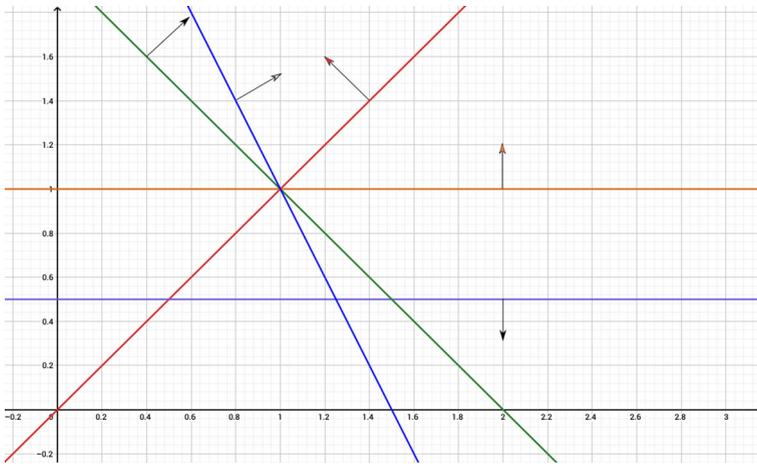


Fig. 1 The set Q

$$x_2 \geq 1, \tag{D}$$

$$2x_1 + x_2 \geq 3. \tag{E}$$

It is easy to check that the following sets are IIS with respect to Q :

$$P_1 = \{(A), (B), (C)\},$$

$$P_2 = \{(A), (B), (E)\},$$

$$P_3 = \{(B), (D)\}.$$

In the continuous case with linear constraints, the work [11] provides a method to identify IIS that amounts to solve a linear problem. For other classes of problems, e.g. a nonlinear system of inequalities, specialized methods and approximations exist, and we refer the reader to the monograph [9] for details.

2.3 IIS and CCP problems

In [21], in order to strengthen the DEP formulation (2) the authors generated cuts based on IIS. In their paper, the set \mathcal{X} was continuous and they used the results of [11] to derive the cuts. To understand their approach, let us consider that the set Q of all S scenario constraints form an infeasible system. Of course this does not mean that the problem itself is infeasible: by removing scenarios such that sum of the respective probabilities remains less or equal to α , the problem must become feasible at some point (otherwise it was infeasible originally). The difficulty lies in choosing the correct constraints to be removed, that is, the constraints that lead both to feasibility and to the maximization of the objective function.

The first step is to consider a simplified version of problem (2), called the pure scenario problem (PSP), obtained by removing the knapsack constraint in problem (2):

$$\begin{aligned}
 \text{PSP: } & \min c'x \\
 \text{s.t. } & Ax \geq b, \\
 & T(\omega)x \geq h(\omega), \quad \forall \omega \in \Omega, \\
 & x \in \mathcal{X}.
 \end{aligned} \tag{3}$$

If PSP is infeasible, by using [11] one can find an IIS, which naturally generates a cut for the DEP. For instance if constraints corresponding to scenarios 1, 4 and 7 form an IIS, the corresponding constraint, or cut, to be added to the DEP is

$$z_1 + z_4 + z_7 \geq 1.$$

Such constraint forces one of the scenarios that belong to this IIS to be left out of the problem. In [21] the authors propose a branch-and-cut algorithm on the variables z that searches for the best scenarios to be removed in the CCP. They report excellent computational results for an extensive set of instances.

It is not necessary to have an infeasible set of scenario constraints to justify the use of chance constraints. It may well be the case that the problem is feasible if all scenarios are included, and the purpose of the chance constraint is to improve the objective function value by removing some scenarios. In this situation, which refer to as the feasible case, it is not clear how IIS can be useful, since there is no infeasibility in the set of constraints. However, when we modify the PSP to include a constraint in the objective function value, infeasibility appears as a way of improving the quality of the solution. In other words, the question in this context is which constraints should be removed in order to improve the objective function value by at least some pre-specified amount. We refer to this formulation as PSPwO, and it can be written as follows:

$$\begin{aligned}
 \text{PSPwO: } & \min c'x \\
 \text{s.t. } & Ax \geq b, \\
 & T(\omega)x \geq h(\omega), \quad \forall \omega \in \Omega, \\
 & c'x \leq u - \epsilon, \\
 & z_\omega \in \{0, 1\}^S, \quad x \in \mathcal{X},
 \end{aligned} \tag{4}$$

where u is an upper bound on the objective function value, obtained for example when solving the problem with all scenarios, and ϵ is an arbitrary value that defines the desired decrease in terms of objective function value. If the problem is infeasible it means that it is not possible to improve on the current bound *with all scenarios in place*. In this case an IIS is found, which necessarily contains the last constraint in PSPwO, and also other scenario constraints. As in the PSP problem, a cut can be constructed such that at least one of the constraints stopping the improvement in terms of objective function value has to be removed.

As we will see later, the choice of ϵ is extremely important in the performance of the algorithm. It is tempting to choose large values of ϵ in order to obtain quick improvements, but if the desired decrease is too large one may have to remove more constraints than allowed by α , which works as a budget on how many constraints can

be removed. On the other hand, small values of ϵ may be too conservative, slowing down the computations. In [21] the authors do not report the value they worked with, and no experiments are performed comparing the performance of different choices. We will discuss in detail those issues in the numerical experiments, and propose a scheme to choose the optimal ϵ in some sense.

For the binary case, that is, when $\mathcal{X} = \{0, 1\}^N$, we are not aware of a result as simple as the one devised in [11] to identify IIS. It is not the purpose of this work to propose a method or an heuristic to obtain IIS for binary problems, so whenever we need to generate one we use in-built functions of commercial solvers for that purpose. We tried to obtain more details of how those functions work, but there is very little information in the solvers' websites. Therefore, we use the functions as blackboxes that generate IIS whenever needed.

3 Strategies to solve binary CCP

In the previous section we described IIS and how they can be used to solve CCP, following the ideas described in [21]. In addition, by using in-built functions of commercial solvers one can in principle address the binary case. In this section we describe two fundamental ideas that make use of the structure of CCP, and of the fact that we are dealing with binary variables.

3.1 Efficient upper bound generation

In every algorithm it is important to find upper bounds mainly for two reasons: to keep track of the best candidate as the algorithm advances, and to have a stopping criteria for termination. It is often the case that obtaining an upper bound is time consuming, and computing it at every iteration can slow down the algorithm significantly. Whenever the problem was feasible in [21], the authors used the PSP problem to obtain upper bounds. In the case where the PSP is feasible, the problem remains feasible as scenarios are removed. In this case, *as long as the sum of the probabilities of the scenarios removed is less or equal to α* , the solution of the PSP problem without those scenarios is feasible to the original problem (1).

We tried a similar approach for the binary case and the results were not satisfactory. The reason is that in our case PSP is a binary problem, and it is more demanding to solve than its continuous counterpart. In most cases, the time taken to compute upper bounds represented as much as 80% of computation time, which is not desirable for larger instances. We simply could not compute upper bounds as often as in the continuous case, but we still needed good upper bounds for fathoming nodes in our algorithm, and for closing the gap quickly. Our approach to deal with this problem is based on the following proposition:

Proposition 1 Consider subsets P and Q of Ω such that $P \subset Q \subset \Omega$, and that

$$\sum_{\omega \in \Omega \setminus P} p_s \leq \alpha,$$

that is, the sum of the probabilities of the elements that do not belong to P is less or equal to the reliability level α . For a set $A \subseteq \Omega$, let x_A be an optimal solution and v_A the optimal value of problem (3) restricted to constraints defined by A . Assuming PSP is feasible when all scenarios are present, we have that

1. The solution x_Ω of PSP with all scenarios, is feasible for the original problem, and the optimal value v_Ω is an upper bound for the true optimal value v^* .
2. The solutions x_P and x_Q are feasible to CCP, and therefore the optimal values v_P and v_Q are upper bounds for v^* .
3. The optimal values of the original problem and of the PSP problem based on sets P , Q and Ω satisfy

$$v^* \leq v_P \leq v_Q \leq v_\Omega.$$

- Proof** 1. As PSP is feasible when all scenarios are present, the optimal solution x_Ω of PSP is feasible to the original problem (2) when we fix $z_\omega = 0, \forall \omega \in \Omega$. Therefore $v_\Omega \geq v^*$.
2. For the set P , the optimal solution x_P of the PSP problem with P scenarios could violate the remaining scenarios constraints. However, by hypothesis the sum of the probability of scenarios belonging to $\Omega \setminus P$ is less or equal to α . Therefore, by fixing $z_\omega = 1$ for those scenarios we have a feasible solution to problem (2). The proof for Q is immediate since $P \subset Q$.
3. Since $P \subset Q \subset \Omega$, the result follows from 1 and 2. \square

Inspired by Proposition 1, our idea is to compute upper bounds only when the *budget* α is fully used, that is, only when one has removed as many scenarios as possible from PSP. For instance, suppose we have 100 scenarios, each one with the same probability, and $\alpha = .05$. This means five scenarios can be removed. We will only solve PSP when five constraints are removed from the set Ω : upper bounds obtained with a smaller number of constraints removed are likely to be worse than the ones obtained when the budget is completely used.

3.2 The choice of ϵ for binary problems

In the feasible case, in order to make use of IIS we add a constraint to PSP that accounts for solution quality. When the problem is feasible, the difficulty is not in finding a feasible solution after a constraint is removed, since any removal will generate a feasible solution because the feasible set is being enlarged. The problem is to choose the constraints to be removed such that the objective function value will decrease.

The value of ϵ should be such that all vectors which generate a decrease of *at least* ϵ in the objective function should be considered. In principle, one would have to use a very small ϵ , e.g. 10^{-5} , in order to avoid excluding candidates that may be optimal for the original problem. However, such small values of ϵ slow the algorithm down considerably, as improvements in the upper bound at every step are marginal.

In the discrete case more can be said about the choice of ϵ . The following proposition assures the existence of an optimal ϵ :

Proposition 2 *Let x^* be an optimal solution of problem (2), and \bar{x} be a feasible solution for the problem such that $u = c'\bar{x} > c'x^*$. Then, there exists $\epsilon > 0$ that does not eliminate any vectors from problem (4) that improve the objective function value and satisfy $Ax \geq b$.*

Proof Let $\mathcal{W} = \{x | Ax \geq b, c'x < u\}$. By hypothesis, the set \mathcal{W} is nonempty: the optimal solution $x^* \in \mathcal{W}$. We define ϵ as follows:

$$\epsilon = \min_{x \in \mathcal{W}} \max\{u - c'x, 0\}. \tag{5}$$

Since our feasible set is finite (and \mathcal{W} is nonempty), the minimum in problem (5) will be attained, leading to a positive value of ϵ . Moreover, by the construction of the set \mathcal{W} , all vectors that improve on the value of u and satisfy $Ax \geq b$ are considered. \square

Unlike the continuous case, Proposition 2 gives a constructive way of finding a value of ϵ that guarantees a decrease in terms of objective function value *without excluding any potential candidate*. However, the result is not practical to implement since we would have to enumerate all vectors x that satisfy $Ax \geq b$. Nevertheless, inspired by the constructive proof of Proposition 2, we propose solving an alternative problem to find a value of ϵ :

$$\begin{aligned} \min_{\epsilon, x} \quad & \epsilon \\ \text{s.t.} \quad & \epsilon \geq u - c'x, \\ & \epsilon \geq 0, \\ & u - c'x \geq \delta, \\ & Ax \geq b, \\ & x \in \{0, 1\}^n, \end{aligned} \tag{6}$$

where $\delta > 0$ is a small enough number that serves the purpose of forcing the problem to move away from the current upper bound. This formulation allows us to find the smallest improvement (that is greater than δ) over the current upper bound u without having to enumerate all possible solutions. We will see that in practice the value of ϵ obtained by solving problem (6) provides excellent guidance for our numerical scheme.

In formulation (6) we are exploiting the fact that our decision variables x are binary. If the decision variables were integers problem (6) becomes challenging and it would slow down our numerical procedure significantly. Since binary variables are bounded, off-the-shelf solvers can quickly find a solution to problem (6).

4 The algorithm

In this section, we present our numerical scheme that uses IIS to solve binary CCP. First, we will define some auxiliary formulations that correspond to intermediate steps

in the procedure, and then we present the step-by-step description of the algorithm. Finally, we show a proof of convergence.

4.1 Preliminaries

Let \mathcal{N} be the set of open nodes indexed by k in a branch-and-bound (BAB) tree on the z_ω variables, $\omega \in \Omega$. Let a path from node k to the root node of the BAB tree be denoted by $\tau(k)$. We define \mathcal{U}^k as the set of scenarios associated with nodes in $\tau(k)$ such that $z_\omega = 1$, and $p(\mathcal{U}^k) \leq \alpha$. Similarly, define $\mathcal{L}^k \subseteq \Omega$ as the set of scenarios associated with nodes in $\tau(k)$ such that $z_\omega = 0$. The following formulation is similar the PSP problem, but it is restricted to scenarios that have not been removed yet:

$$\begin{aligned}
 \text{PSP2}^k : \min \quad & c'x \\
 \text{s.t.} \quad & Ax \geq b, \\
 & T(\omega)x \geq h(\omega), \quad \forall \omega \in \Omega \setminus \mathcal{U}^k, \\
 & x \in \mathbb{B}^n.
 \end{aligned} \tag{7}$$

As stated in Sect. 2.3, if (7) is feasible for $\mathcal{U}^k = \emptyset$ (i.e. the root node) then we can add to PSP2^k the constraint $c'x \leq u - \epsilon$, where u is the current upper bound. Now let S_j be an IIS of (7) and $D_j := \{\omega \in \Omega : T(\omega)x \geq h(\omega) \cap S_j \neq \emptyset\}$. If (7) is infeasible, we can obtain an IIS S_j that determines the set D_j . Then the IIS inequality $\sum_{\omega \in D_j} z_\omega \geq 1$ is valid for (2) by construction. Given the sets \mathcal{U}^k and \mathcal{L}^k , the problem to solve at node k is:

$$\begin{aligned}
 \text{PSP3}^k : \min \quad & c'x \\
 \text{s.t.} \quad & Ax \geq b, \\
 & T(\omega)x + Mz_\omega \geq h(\omega), \quad \forall \omega \in \Omega \setminus \{\mathcal{U}^k \cup \mathcal{L}^k\}, \\
 & T(\omega)x \geq h(\omega), \quad \forall \omega \in \mathcal{L}^k, \\
 & \sum_{\omega \in \Omega \setminus \{\mathcal{U}^k \cup \mathcal{L}^k\}} p_\omega z_\omega \leq \alpha - \sum_{\omega \in \mathcal{U}^k} p_\omega z_\omega, \\
 & \sum_{\omega \in D_j} z_\omega \geq 1, \quad \forall j \in \tau(k), \\
 & x \in \mathbb{B}^n, z \in \mathbb{B}^S.
 \end{aligned} \tag{8a}$$

By adding an IIS inequality (8b) at a particular node of the BAB tree, at least one scenario is *excluded* from the nodal problem so that the total number of nodes to search in the BAB tree is reduced. Moreover, we note that constraint (8a) is updated based on scenarios already removed in the scenario tree, that is, belonging to the set \mathcal{U}^k .

4.2 IIS branch-and-cut algorithm

A detailed description is given in Algorithm 1.

Algorithm 1 IIS branch-and-cut algorithm (IISBAC)

- 1: **Initialize:** Set $\mathcal{L}^1 = \emptyset, \mathcal{U}^1 = \emptyset, n^1 = (\mathcal{L}^1, \mathcal{U}^1), \mathcal{N}^1 = \{n^1\}, K = 1, u = \infty, l = -\infty$.
 - 2: **Node Choice:** Pick some node $n^k \in \mathcal{N}$ according to some search rule. If $\mathcal{N} = \emptyset$, then terminate execution.
 - 3: **Solve LP:** Solve (8) relaxing the integrality on the z variables. This will either find an optimal solution $(\bar{x}, \{\bar{z}_\omega\}_{\omega \in \Omega})$, or that the problem is *infeasible*.
 - 4: **Fathoming Rule:** If the node relaxation is infeasible, or $c'x \geq u$, then fathom the node and return to 1. Otherwise, solve (7). If (7) is feasible, then set $z_\omega = 0, \forall \omega \in \Omega \setminus \mathcal{U}^k$ obtaining a feasible integer incumbent, update the upper bound and go to 2. Otherwise go to 5.
 - 5: **IIS Cut Generation:** Obtain the set D_j using some IIS search method. If $D_j \neq \emptyset$, then add the cut $\sum_{\omega \in D_j} z_\omega \geq 1$ and go to 3. Otherwise, improve (if possible) the upper bound u and go to 6.
 - 6: **Branching:** Pick a non-integer z variable using some branching rule. Create new nodes $n^{k+1} = (\mathcal{L}^k \cup z_\omega, \mathcal{U}^k)$ and $n^{k+2} = (\mathcal{L}^k, \mathcal{U}^k \cup z_\omega)$. Add these nodes to \mathcal{N} , set $k = k + 2$ and go to 3.
-

In our implementation, we used Proposition 1 to improve the algorithm in step 4. Given a set \mathcal{U}^k , we define $Q = \Omega \setminus \mathcal{U}^k$. By removing additional scenarios from Ω and adding them to set \mathcal{U}^k such that the budget α is not exceeded, we construct the set $\mathcal{U}^{k'} \supset \mathcal{U}^k$, and define $P = \Omega \setminus \mathcal{U}^{k'} \subset Q$. Following the third claim of Proposition 1, the upper bound obtained using the set P is less than or equal to the one based on set Q . It is important to highlight that such scheme is only valid for the computation of the upper bound: the cut is always constructed based on the original set \mathcal{U}^k .

Our algorithm works like a standard branch-and-cut algorithm, as we are only adding an intermediary step in order to add our cuts using the information obtained by the IIS. This algorithm still relies on finding lower bound (improved by our cuts) and upper bounds (improved by our checking step, when possible), and closing the gap between them as the termination criterion.

Fathoming nodes is important to speed up the process of finding the optimal solution. We have two rules: fathoming by optimality or infeasibility. The former can be applied by checking the upper bound (step 2), the latter by not being able to find an IIS that relies on stochastic constraints (i.e. all IIS members are rows of the A matrix). Termination is guaranteed by the following proposition.

Proposition 3 *The IIS branch-and-cut algorithm terminates in a finite number of iterations and finds the optimal solution if it exists.*

Proof In order to prove finiteness, consider we are solving our problem (2) with $n + S$ binary variables, then we have 2^{n+S} possible combinations of solutions, thus in the worst case we might explore all possible solutions, which are finite.

Now, we must prove that the algorithm does not remove the optimal solution. As we have discussed before: there are two main cases to consider. If (3) is infeasible in the root node, and assuming (2) has an optimal solution, the algorithm focuses in cutting infeasible leaves, which are a subset of the 2^{n+S} possible combinations. Therefore, we are reducing the number of combinations to explore, while avoiding cutting the

optimal solution. Since we are solving (8), we are always checking feasibility in the knapsack, then the solution will be feasible (2).

There is another case to consider: if problem (3) is feasible at the root node. While we select an $\epsilon > 0$, the algorithm cuts feasible leaves of solution combinations and by construction the upper bound improves each time, since we will fathom all nodes that have a worst bound than the incumbent upper bound. This means we are not cutting nodes which might contain the optimal solution, since they have solutions which yield values strictly smaller than our current upper bound. Henceforth, we are reducing the number of nodes to explore. \square

As mentioned, obtaining an IIS for binary problems is involved, and we use function `ComputeIIS()` of the `model` class provided by the solver Gurobi. As mentioned, our focus is not on finding a method to obtain an IIS with binary variables, instead we concentrate on obtaining valid cuts using an IIS. Unfortunately, there is no documentation about *how* this function works, and we have no control over it.

5 Numerical experiments

We present two sets of experiments: first, consider a joint chance-constrained version of the probabilistic set-covering (PSC) problem, as described in [5]. In the problem matrix T has dimensions $m_2 \times n$, $m_2 > 1$. This means that we have a set of m_2 constraints inside the chance-constraint. The only source of randomness in this problem is vector $h(\omega)$, with dimensions $m_2 \times 1$. Furthermore, the coefficients of the objective function and the entries of matrix T are binary. As we will show, we need to pre-process the problem in order to solve it efficiently. In addition, writing the cut derived by the IIS is more challenging in this situation because we have joint chance-constraints. The z_ω variables have a different meaning in this case: whenever z_ω is equal to one, a whole block of m_2 constraints is removed, rather than a single constraint. Since the IIS oracle returns constraints, it is not clear how to move from constraints to blocks for IIS cut construction. We propose a rule that tells us how many and which blocks to remove given the set of constraints belonging to the IIS, and tested our approach on several instances of the PSC problem.

The second experiment is a vaccine allocation problem, where $T(\omega)$ is a $1 \times n$ matrix and $h(\omega) \in \mathbb{R}$, for all $\omega \in \Omega$. In this case, the application of IIS cuts is direct: whenever a constraint belongs to an IIS, it is added as an element in the cut. The coefficients in the objective function are non-integers, and we will show that it is very important in this case to choose ϵ appropriately. This problem was analyzed in [21] for continuous variables.

The objective of our experiments was to validate the methodology and to reduce the number nodes explored in the IISBAC algorithm compared to a standard solver. In this sense, we want to compare number of nodes explored, which does not depend on the programming language used to implement the algorithm. For the sake of completeness, we also compare running times.

In our experiments we used an implementation of IISBAC algorithm in Python 3.6.2 using Gurobi 7.5.1. The experiments were run on an Intel(R) Xeon(R)

CPU E5-2670 @ 2.60GHz (using 8 threads), 32GB of memory and running CentOS 6.8.

5.1 Stochastic set covering problem

The formulation for this problem as described in [5] is given as follows:

$$\begin{aligned}
 \text{PSC: } \min \quad & c'x \\
 \text{s.t. } \quad & Tx + z_\omega \geq h(\omega), \quad \forall \omega \in \Omega, \\
 & \sum_{\omega \in \Omega} z_\omega \leq \lfloor S\alpha \rfloor, \\
 & z_\omega \in \mathbb{B}^S, \quad x \in \mathbb{B}^n.
 \end{aligned} \tag{9}$$

Problem (9) is a joint chance-constrained problem, so each scenario ω corresponds to a block of constraints. The difficulty here is that the IIS routine returns the constraints to be removed, while we need to decide on the blocks to be removed. The cuts based on IIS would still be valid, and would eventually reach the optimal solution. However, we could not reach the optimal solution for most instances. We realized that in this particular case, we could use a heuristic: let the *cut length* l be the amount of blocks that will belong to the cut (i.e. the amount of z_ω variables present in the cut). So, by bounding the cut length to a fixed number, we will only include the first l blocks selected by sorting them in nondecreasing order of the number of *constraints* that are part of the IIS, which is the output given by the solver. For instance assume $l = 2$ and we have three blocks of constraints, each with eight constraints ordered accordingly. If constraints 1, 9, 12, 15, 18 and 22 are part of an IIS, we would add the cut $z_2 + z_3 \geq 1$ because the first block has only one constraint belonging to the IIS (constraint 1), and blocks 2 and 3 have three (9, 12 and 15) and two constraints (18 and 22), respectively.

5.1.1 Results

There was a preprocess step used to reduce the number of stochastic constraints in this problem. We use $\alpha = 0.1$ for this problem, and for instance in the case of 1000 samples our removal budget is equal to 100. Since only the right hand side is random, whenever we have more than 100 copies of a constraint the budget is not enough to eliminate of such constraint. In this case, since the constraint will have to be satisfied, we added it as a row of the system $Ax \geq b$. Using this simple technique, which surprisingly was not done automatically by Gurobi, allowed the solver to handle the instances considered in the experiment.

Table 1 shows the performance of both Gurobi and IISBAC using different cut lengths for 200 replications, with sample sizes ranging from 100 until 10,000. The mean value of ϵ was 1, with a variance at most $O(10^{-10})$ among the replications. To understand the effect of the cut length parameter l , we used values between 1 and 50. It can be seen that as l increases, the values obtained by IISBAC converge to the optimal value found by Gurobi. This requires exploring a larger number of nodes, and

Table 1 Comparison of performance between Gurobi and IISBAC using different sample sizes in `scp41`

Sample size	Average	Gurobi	IISBAC Cut length l					
			1	2	5	10	20	50
100	Nodes	2.5	2.7	2.8	2.7	2.7	2.8	4.3
	Obj. val.	377.0	378.4	378.1	377.8	377.6	377.3	377.0
	Cuts	–	0.1	0.1	0.2	0.2	0.4	0.0
	Time (s)	0.1	6.2	7.8	9.5	13.7	18.2	32.9
1000	Nodes	1542.3	927.0	3092.0	3142.5	3551.7	2661.0	2089.6
	Obj. val.	382.9	383.8	383.4	383.1	383.0	383.1	382.9
	Cuts	–	3.5	4.6	5.3	6.2	6.9	0.0
	Time (s)	17.0	50.2	101.2	100.4	111.4	109.2	130.3
5000	Nodes	921.0	1043.9	1937.9	1401.7	1537.3	1774.7	1254.0
	Obj. val.	381.1	381.3	381.2	381.2	381.1	381.1	381.1
	Cuts	–	2.7	2.8	3.6	3.4	3.0	0.0
	Time (s)	168.5	247.8	380.1	344.4	398.4	393.2	264.9
10,000	Nodes	3.3	8.7	6.5	11.8	12.4	11.6	7.6
	Obj. val.	380.1	380.2	380.1	380.1	380.1	380.1	380.1
	Cuts	–	2.3	2.4	3.2	3.3	3.3	0.0
	Time (s)	90.5	144.2	146.1	156.9	163.1	176.0	176.5

taking more time on average. The PSC `scp41` from [4] is a joint chance-constrained problem with blocks of size 200, which makes it extremely challenging to solve it. Our main purpose was to show that IIS can handle this problem if l is sufficiently large, and that the computational times and nodes explored remain in the same order of magnitude as Gurobi's.

We believe significant time reductions can be achieved by using more sophisticated implementations of the IISBAC. Regarding nodes explored, Gurobi identifies the structure of the PSC problem and uses sophisticated cuts in order to explore less nodes. It seems that for other joint chance-constrained problems IIS cuts could have a significant impact, and could reduce the total numbers explored by a significant amount. The heuristic shows promising results for the joint chance-constrained case, but further research is needed to fully test more criteria in terms of choosing a cut length, or the prioritization to select which scenarios to choose for the cut. Our findings show that if optimality is not critical, very good solutions (within 1% of the optimal) can be found quicker than Gurobi by using smaller cut lengths. By finding good candidates quicker, we could use this approach to give the solver a good starting point. If optimality is critical, then larger cut lengths are needed, possibly combined with another cutting generation procedure to speed up the method.

In order to gain a deeper understanding of our approach, we also applied this methodology for PSC problems `scpe1` (blocks of size 50) and `scpa1` (blocks of size 300). For `scpe1` IISBAC found the optimal solution when using 1000, 5000 and 10,000 samples in less than 600 s. Moreover, for those smaller instances the compu-

Table 2 Comparison of performance between Gurobi and IISBAC using different sample sizes in *scca1*

Sample size	Average	Gurobi	IISBAC cut length <i>l</i>					
			1	2	5	10	20	50
1000	Nodes	622.54	436.97	396.94	417.72	449.81	467.06	454.87
	Obj. val.	232.16	233.89	233.74	233.54	233.34	233.24	233.05
	Cuts	–	4.34	4.29	4.53	4.26	4.69	4.94
	Time (s)	44.09	6345.88	6486.21	6321.03	5545.52	6174.03	5928.11
5000	Nodes	346.14	83.63	84.075	84.37	85.25	86.4	89.16
	Obj. val.	232.4	234.43	234.28	234.18	234.04	233.87	233.68
	Cuts	–	4.34	4.22	4.69	4.07	4.54	4.32
	Time (s)	725.01	6379.05	6450.75	6998.06	6754.95	6844.34	6919.18

tations were not sensitive to the cut length, and the algorithm visited less nodes than Gurobi.

Table 2 describes the results for *scca1*, and in this case a time limit of 7200 s was imposed given the size of the problem. Results show that IISBAC obtained solutions within 2% of the optimal, and the number of nodes explored is smaller than Gurobi’s. The effect is more salient as the sample size increases, showing the potential of IISBAC for solving linear large-scale joint chance-constrained problems.

5.2 Vaccine allocation problem

The formulation for this problem is:

$$\begin{aligned}
 \text{VAC: } & \min c'x \\
 \text{s.t. } & Ax = 1, \\
 & T(\omega)x + M_\omega z_\omega \geq 1, \quad \forall \omega \in \Omega, \\
 & \sum_{s \in \Omega} z_\omega \leq \lfloor S\alpha \rfloor, \\
 & z_\omega \in \mathbb{B}^S, \quad x \in \mathbb{B}^n.
 \end{aligned} \tag{10}$$

The objective is to minimize the cost of assigning vaccines between families in a given sector (given by the *A* matrix), while bounding the probability of spreading the contagion between these families (given by the *T* matrix). In this case, it was needed to use a SAA approach to obtain samples of size *S* from the random variable. As stated before, for every $\omega \in \Omega$ we have that *T*(ω) represents just one constraint.

Table 3 presents the problem sizes for the test instances. As can be seen, the number of decision variables stays constant, equal to $n = 302$, and the number of rows in the *A* matrix is fixed in 30. Nonetheless, the number of scenario variables (and rows) increases depending of the sample used. Therefore, *S* ranges from 100 until 2000. As stated in [21], these instances are difficult to solve because the MIP formulations are

Table 3 Problem sizes for vaccination test instances

Instance	Rows	Decision vars.	Scenario vars.	Knapsack budget
vac100	131	302	100	5
vac250	281	302	250	12
vac500	531	302	500	25
vac750	781	302	750	37
vac1000	1031	302	1000	50
vac2000	2031	302	2000	100

* $\alpha = 0.05$

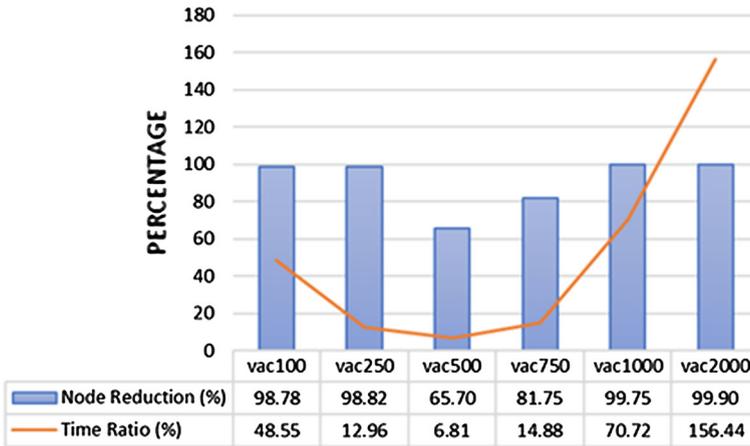


Fig. 2 Performance comparison between instances

large and dense. We included a last column which describes the number of scenarios that can be violated at the same time in an incumbent solution (we consistently used the value $\alpha = 0.05$).

5.2.1 Results

First we present a summary of our findings, and then we highlight some salient features of our approach. Figure 2 shows a comparison between the performance of Gurobi and IISBAC algorithm throughout the different instance sizes. The node reduction, defined as the ratio of nodes explored between Gurobi and IISBAC was on average 90.78%, and on larger instances it was over 99%. This is a remarkable difference with respect to the joint case, which indicates that individual constraints, or possible joint chance-constraints with moderate values of m_2 , could benefit from our approach.

In terms of time, Gurobi is able to outperform IISBAC on small instances, but the solution times are comparable when we used 1000 samples. Even though we are using a language suitable for prototyping, our implementation was faster than Gurobi for the larger 2000-sample instances. Our explanation is that as the problem becomes larger, more effort is necessary to solve the problem as new constraints and binary variables

Table 4 Larger samples (1000 and 2000) results using $\alpha = 0.05$

Sample	Gurobi	IISBAC	Gurobi	IISBAC	IISBAC info	
	Time		Nodes		Node red. %	Cuts
vac1000a	2936	4344	144,904	2910	97.99	169
vac1000b	1000	3229	156,803	2362	98.49	213
vac1000c	3449	10,045	1,056,888	1275	99.88	168
vac1000d	6488	2943	1,945,060	1365	99.93	176
vac1000e	2437	2503	546,318	1536	99.72	9
vac2000a	51,456	14,980	3,335,160	3438	99.90	255
vac2000b	30,218	83,717	1,540,281	5827	99.62	474
vac2000c	> 100,000*	66,066	> 12,184,026*	9325	99.92	661
vac2000d	84,524	24,043	4,555,042	3626	99.92	310
vac2000e	69,259	25,623	6,034,681	4127	99.93	292

* Final gap 1.5%

are being added. Thus, as complexity rises, any help to reduce said effort is going to have a significant impact on the solver's performance.

We ran more tests with different values of ϵ in this formulation, since it is easier to solve than the PSC. We were consistently lowering the number of nodes explored, which was our objective, but we noticed that the variance of the ϵ 's calculated by (6) was at most $O(10^{-2})$ with mean 0.3076. The practical conclusion is that problem (6) is essential to the performance of the algorithm, but it does not need to be solved more than once.

To stress this fact, we note that using different α values of ϵ will produce very different outcomes: if $\epsilon < 0.3076$, IISBAC will produce smaller improvements on the upper and lower bounds, which in turn means more nodes to explore. We were able to detect that for values close to 0.001, the amount of nodes can quickly surpass Gurobi's count. On the other hand, if $\epsilon > 0.3076$ then the optimal solution could potentially be cut from the tree.

Table 4 zooms in on larger instances considered for this problem, consisting of 1000 and 2000 samples. On average, the IISBAC algorithm explored between 0.01 and 2% of the total nodes of Gurobi, with less computational time in half of the instances. We detected that the first node added was able to improve the upper bound obtained by Gurobi up to 0.1% over the optimal value in every case (between the first 3 min of running time), while the rest of the time was spent to close the gap by improving the lower bound via cuts.

Finally, the second half of Table 4 shows promising results. IISBAC explores less than 1% of the nodes that Gurobi explores on average, while doing so in less time than the solver: we can observe solving times being around 25% those of the solver. Note that instance *vac2000c* using Gurobi was only able to reach a gap of 1.5% after 600,000 s (almost 7 days), when it was decided to interrupt the computations.

As can be seen with the vaccine problem, the IISBAC algorithm is able to reduce the effort in solving this problem, and as the amount of samples used to solve the

problem increases, so does the effectiveness of the IIS cuts applied. We were able to observe that while the solver works on the root node of the branch-and-bound tree, the first nodes added by IISBAC algorithm are able to obtain on average an upper bound within 1% of the optimal value, while the rest of the time was an effort to close the lower bound, using the cuts.

It is important to highlight that when α is greater than or equal to 0.1 the problem becomes more complex. The combinatorics for the possible ways of choosing which constraints can be violated increases, and both Gurobi and our algorithm provide unsatisfactory solutions. For $\alpha = .1$ Gurobi cannot reduce the optimality gap to less than 5% after 2 h in any of the 1000 and 2000 samples instances. The same happened with IISBAC: the gap was around 20% in most instances under the same time limit, while the number of nodes explored was around .02% of the nodes explored by Gurobi. In personal communications with a senior developer of Gurobi we concluded that this vaccine problem is especially challenging because some of the coefficients are of small magnitude small compared to the objective function value coefficients. We believe that for other linear binary binary chance-constrained problems IISBAC would perform better for values of α greater or equal to 10%.

6 Conclusions

In this paper, we propose a novel methodology to solve binary chance-constrained problems building on [21]. In particular, we develop improvements for finding upper and lower bounds by using the structure of the problem to our advantage. A salient feature of our approach is the idea of adding a constraint that can improve on the quality of the solution at every iteration, which allows us to handle cases where the problem with all scenarios in place is feasible. We solve an auxiliary problem that allows to define the maximum possible improvement at every iteration, and the efficiency depends directly on the fact that our decision vector is binary.

The algorithm can handle joint and individual linear CCP, and we tested our approach on a probabilistic version of the set covering problem, and on a vaccine allocation problem. In the first case, a joint CCP, some extra work is required to convert the output of the IIS generator to cuts that remove blocks of constraints, and a heuristic was implemented to handle the priority in which these blocks should be added. Computational results, on problems with 50, 200 and 300 constraints per scenario (we considered up to 10,000 scenarios) indicate that the method finds the optimal solution for cuts with length 50, and finds solutions within 1% of the optimal for smaller cut lengths, with less computational effort. For individual CCP computational results show that in many instances we reduced drastically the number of nodes explored, and in some cases the computational time was comparable to Gurobi even though we did not aim at maximum computational efficiency.

The main goal of the paper was to establish IIS as a tool to solve binary CCP. For the joint case more research is needed to reduce computational times, and for the individual case the results indicate that significant gains in terms of speed can be achieved using our methodology. A natural extension of the current work would start with problems with a single chance-constraint, and move gradually to more complex joint CCP, in

order to gain a better understanding of the impact in performance of moving from constraint removal to block removal. For higher values of the reliability level α the results were unsatisfactory, both with Gurobi and our algorithm. We believe it is due to the small magnitude of some coefficients in the constraints with respect to the constants in the objective function. Further research is needed in order to be able to cope with the combinatorial explosion that occurs when more constraints are allowed to be violated. On the applications side, binary CCP arise often in natural resource management problems, in particular in mining applications where the decision variables are whether or not to remove a block from the mine. Uncertainty is present via ore prices or grade distribution, and the objective is to maximize net present value of the operations, a description that fits into the framework developed in this work.

References

1. Abdelaziz, F.B., Aouni, B., El Fayedh, R.: Multi-objective stochastic programming for portfolio selection. *Eur. J. Oper. Res.* **177**(3), 1811–1823 (2007)
2. Abdi, A., Fukasawa, R.: On the mixing set with a knapsack constraint. *Math. Program.* **157**(1), 191–217 (2016)
3. Ahmed, S., Papageorgiou, D.J.: Probabilistic set covering with correlations. *Oper. Res.* **61**(2), 438–452 (2013)
4. Beasley, J.E.: An algorithm for set covering problem. *Eur. J. Oper. Res.* **31**(1), 85–93 (1987)
5. Beraldi, P., Ruszczyński, A.: The probabilistic set-covering problem. *Oper. Res.* **50**(6), 956–967 (2002)
6. Campi, M.C., Garatti, S.: The exact feasibility of randomized solutions of uncertain convex programs. *SIAM J. Optimiz.* **19**(3), 1211–1230 (2008)
7. Campi, M.C., Garatti, S., Prandini, M.: The scenario approach for systems and control design. *Annu. Rev. Control* **33**(2), 149–157 (2009)
8. Charnes, A., Cooper, W.W., Symonds, G.H.: Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. *Manag. Sci.* **4**(3), 235–263 (1958)
9. Chinneck, J.W.: Feasibility and infeasibility in optimization: algorithms and computational methods, vol. 118. Springer, Berlin (2007)
10. Dentcheva, D., Prékopa, A., Ruszczyński, A.: Concavity and efficient points of discrete distributions in probabilistic programming. *Math. Program.* **89**(1), 55–77 (2000)
11. Gleeson, J., Ryan, J.: Identifying minimally infeasible subsystems of inequalities. *ORSA J. Comput.* **2**(1), 61–63 (1990)
12. Küçükyavuz, S.: On mixing sets arising in chance-constrained programming. *Math. Program.* **132**(1–2), 31–56 (2012)
13. Kumral, M.: Application of chance-constrained programming based on multi-objective simulated annealing to solve a mineral blending problem. *Eng. Optimiz.* **35**(6), 661–673 (2003)
14. Lejeune, M., Noyan, N.: Mathematical programming approaches for generating p-efficient points. *Eur. J. Oper. Res.* **207**(2), 590–600 (2010)
15. Lejeune, M.A.: Pattern-based modeling and solution of probabilistically constrained optimization problems. *Oper. Res.* **60**(6), 1356–1372 (2012)
16. Luedtke, J., Ahmed, S.: A sample approximation approach for optimization with probabilistic constraints. *SIAM J. Optimiz.* **19**(2), 674–699 (2008)
17. Pagnoncelli, B., Ahmed, S., Shapiro, A.: Sample average approximation method for chance constrained programming: theory and applications. *J. Optimiz. Theory Appl.* **142**(2), 399–416 (2009)
18. Prékopa, A.: Dual method for the solution of a one-stage stochastic programming problem with random rhs obeying a discrete probability distribution. *Z. Oper. Res.* **34**(6), 441–461 (1990)
19. Saxena, A., Goyal, V., Lejeune, M.A.: Mip reformulations of the probabilistic set covering problem. *Math. Program.* **121**(1), 1–31 (2010)
20. Song, Y., Luedtke, J.R., Küçükyavuz, S.: Chance-constrained binary packing problems. *INFORMS J. Comput.* **26**(4), 735–747 (2014)

21. Tanner, M.W., Ntamo, L.: IIS branch-and-cut for joint chance-constrained stochastic programs and application to optimal vaccine allocation. *Eur. J. Oper. Res.* **207**(1), 290–296 (2010)
22. Zhu, M., Taylor, D.B., Sarin, S.C., Kramer, R., et al.: Chance constrained programming models for risk-based economic and policy analysis of soil conservation. *Agric. Resour. Econ. Rev.* **23**(1), 58–65 (1994)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.