Artificial Intelligence 189 (2012) 69–94

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Filtering algorithms for global chance constraints ☆

Brahim Hnich [a], Roberto Rossi [b,*], S. Armagan Tarim [c], Steven Prestwich [d]

[a] Department of Computer Engineering, Izmir University of Economics, Turkey
[b] University of Edinburgh Business School, 29 Buccleuch Place, EH8 9JS, Edinburgh, UK
[c] Department of Management, Hacettepe University, Ankara, Turkey
[d] Cork Constraint Computation Centre, University College Cork, Ireland

ARTICLE INFO

Article history:
Received 8 June 2011
Received in revised form 7 May 2012
Accepted 10 May 2012
Available online 14 May 2012

Keywords:
Stochastic constraint programming
Stochastic constraint satisfaction
Global chance constraints
Filtering algorithms
Stochastic alldifferent

ABSTRACT

Stochastic Constraint Satisfaction Problems (SCSPs) are a powerful modeling framework for problems under uncertainty. To solve them is a PSPACE task. The only complete solution approach to date — scenario-based stochastic constraint programming — compiles SCSPs down into classical CSPs. This allows the reuse of classical constraint solvers to solve SCSPs, but at the cost of increased space requirements and weak constraint propagation. This paper tries to overcome these drawbacks by automatically synthesizing filtering algorithms for global chance constraints. These filtering algorithms are parameterized by propagators for the deterministic version of the chance constraints. This approach allows the reuse of existing propagators in current constraint solvers and it has the potential to enhance constraint propagation. Our results show that, for the test bed considered in this work, our approach is superior to scenario-based stochastic constraint programming. For these instances, our approach is more scalable, it produces more compact formulations, it is more efficient in terms of run time and more effective in terms of pruning for both stochastic constraint satisfaction and optimization problems.

1. Introduction

In this work we consider problems in which we are required to make decisions under uncertainty. The word *uncertainty* is used to characterize the existence, in these problems, of uncontrollable or "random" variables,[1] which cannot be influenced by the decision maker. In addition to random variables, the problems we consider also comprise controllable or "decision" variables, to which a value from given domains has to be assigned. More specifically, a problem classified as *deterministic* with respect to the degree of uncertainty does not include random variables, while a *stochastic* problem does. Random variables are typically employed to model factors such as the customer demand for a certain product, the crop yield of a given piece of land during a year, the arrival rate of orders at a reservation center and so forth. A continuous or discrete domain of possible values that can be observed is associated with each random variable. A probabilistic measure — typically a probability distribution — over such a domain is assumed to be available in order to fully quantify the likelihood of each value (respectively, range of values in the continuous case) that appears in the domain. The decision making process comprises one or more consecutive *decision stages*. In a decision stage, a decision is taken by the decision

☆ This work is an extended version of Hnich et al. (2009) [11]. S. Armagan Tarim is supported by Hacettepe University (HU-BAB) and the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. 110M500.

* Corresponding author. Tel.: +44 (0) 131 651 4389.
E-mail addresses: brahim.hnich@ieu.edu.tr (B. Hnich), roberto.rossi@ed.ac.uk (R. Rossi), armagan.tarim@hacettepe.edu.tr (S.A. Tarim), s.prestwich@4c.ucc.ie (S. Prestwich).

[1] Alternatively, in the literature, these variables are also denoted as "stochastic".

0004-3702/$ – see front matter
http://dx.doi.org/10.1016/j.artint.2012.05.001

maker who assigns a value to each controllable variable related to this decision stage of the problem and, subsequently, the uncontrollable variables related to this stage are observed and their realized values become known to the decision maker.

Stochastic constraint satisfaction problems (SCSPs) are a powerful modeling framework for decision making under uncertainty. SCSPs were first introduced in [28] and further extended in [26] to allow multiple chance constraints and a range of different objectives. SCSP is a PSPACE-complete problem [28]. The approach in [26] compiles down SCSPs into deterministic equivalent CSPs. Intuitively, the compilation strategy in [26] relies on a heavy use of binary variables that are employed in order to encode every single possible future scenario in a monolithic constraint programming model. This makes it possible to reuse existing solvers, but at the cost of increased space requirements and of weakened constraint propagation.

In this paper we overcome these drawbacks by automatically synthesizing filtering algorithms for global chance constraints. These filtering algorithms are built around propagators for the deterministic version of the chance constraints. Like the approach in [26], our approach reuses the propagators already available for classical CSPs; nevertheless, our approach uses fewer decision variables — since it does not rely on a reformulation employing binary variables associated with scenarios — and it has the potential to strengthen constraint propagation. The ease and power of the generic modeling tools discussed in this paper make our approach appealing. For the test bed considered in this work, our approach is superior to the one in [26], since it is more scalable, it produces more compact formulations, and it achieves stronger pruning; in these experiments our approach is more efficient also in terms of run time and explored nodes for both stochastic constraint satisfaction and optimization problems.

This work extends preliminary results presented in [11]. In particular, we have introduced additional material proving the intractability of maintaining generalized arc consistency for global chance constraints that embed a global constraint for which a poly-time propagator exists. We have introduced two incremental filtering algorithms that were not included in [11]. In addition to the random stochastic constraint satisfaction problems discussed in [11], we have tested out approach on two additional benchmark problems: the static stochastic knapsack problem and the stochastic plane landing scheduling problem. The thorough experimental analysis in this work significantly extends the one in [11].

The paper is structured as follows: in Section 2 we provide the relevant formal background; in Section 3 we discuss the structure of an SCSP solution; in Section 4 we describe the state-of-the-art approach to SCSPs; in Sections 5 and 6 we discuss our novel approach; in Section 7 we propose incremental versions of our filtering algorithm; in Section 8 we discuss our benchmark problems; in Section 9 we present our computational experience; in Section 10 we provide a brief literature review; finally, in Section 11 we draw conclusions and outline our future work.

2. Formal background

A Constraint Satisfaction Problem (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for some variables. A *solution* to a CSP is an assignment of variables to values in their respective domains such that all of the constraints are satisfied. Constraint solvers typically explore partial assignments enforcing a local consistency property. A constraint c is *generalized arc consistent* (*GAC*) if and only if when a variable is assigned any of the values in its domain, there exist compatible values in the domains of all the other variables of c. In order to enforce a local consistency property on a constraint c during search, we employ filtering algorithms that remove inconsistent values from the domains of the variables of c. These filtering algorithms are repeatedly called until no more values are pruned. This process is called *constraint propagation*.

An m-stage SCSP is defined as a 7-tuple $\langle V, S, D, P, C, \theta, L\rangle$, where V is a set of decision variables and S is a set of stochastic variables, D is a function mapping each element of V and each element of S to a domain of potential values. In what follows, we assume that both decision and stochastic variable domains are finite. P is a function mapping each element of S to a probability distribution for its associated domain. C is a set of chance constraints over a non-empty subset of decision variables and a subset of stochastic variables. θ is a function mapping each chance constraint $h \in C$ to θ_h which is a threshold value in the interval $(0, 1]$. $L = [\langle V_1, S_1\rangle, \ldots, \langle V_i, S_i\rangle, \ldots, \langle V_m, S_m\rangle]$ is a list of *decision stages* such that each $V_i \subseteq V$, each $S_i \subseteq S$, the V_i form a partition of V, and the S_i form a partition of S.

The solution of an m-stage SCSP is, in general, represented by means of a *policy tree* [26]. The arcs in such a policy tree represent values observed for stochastic variables whereas nodes at each level represent the decisions associated with the different stages. We call the policy tree of an m-stage SCSP that is a solution a *satisfying policy tree*.

3. Satisfying policy trees

In order to simplify the presentation, we assume without loss of generality that each $V_i = \{x_i\}$ and each $S_i = \{s_i\}$ are singleton sets. All the results can be easily extended in order to consider $|V_i| > 1$ and $|S_i| > 1$. In fact, if S_i comprises more than one random variable, it is always possible to aggregate these variables into a single multivariate random variable [13] by convoluting them. If V_i comprises more than one decision variable, the following discussion still holds, provided that the term *DecVar*, which we will introduce in the next paragraph, is interpreted as a set of decision variables.

Let $S = \{s_1, s_2, \ldots, s_m\}$ be the set of all stochastic variables in the problem and $V = \{x_1, x_2, \ldots, x_m\}$ be the set of all decision variables. In an m-stage SCSP, the policy tree has

$$\mathcal{N} = 1 + |s_1| + |s_1| \cdot |s_2| + \cdots + |s_1| \cdot |s_2| \cdot \cdots \cdot |s_{m-1}| = 1 + \sum_{i=1}^{m-1} \prod_{j=1}^{i} |s_j|$$

nodes, where $|s_j|$ denotes the cardinality of $D(s_j)$. We adopt the following node and arc labeling schemes for the policy tree of an m-stage SCSP. The depth of a node can be uniquely associated with its respective decision stage, more specifically V_i is associated with nodes at depth $i-1$. We label each node with $\langle DecVar, DecVal, Index\rangle$ where $DecVar$ is a decision variable that must be assigned at the decision stage associated with the node, $DecVal \in D(DecVar)$ is the value that this decision variable takes at this node, and $Index \in \{0, \ldots, \mathcal{N}-1\}$ is a unique index for this node. Each arc will be labeled with $\langle StochVar, StochVal\rangle$ where $StochVar \in S$ and $StochVal \in D(StochVar)$. According to our labeling scheme, the root node has label $\langle x_1, \bar{x}_1, 0\rangle$ where $\bar{x}_1$ is the value assigned to the variable x_1 associated with the root node and the index of the root node is 0. The root node is at depth 0. For each value $\bar{s}_1 \in D(s_1)$, we have an arc leaving the root node labeled with $\langle s_1, \bar{s}_1\rangle$. The $|s_1|$ nodes connected to the root node are labeled from 1 to $|s_1|$. For each node at depth 1, we label each of $|s_2|$ arcs with $\langle s_2, \bar{s}_2\rangle$ for each $\bar{s}_2 \in D(s_2)$. For the nodes at depth 2, we label them from $\langle x_2, \bar{x}_2, |s_1|+1\rangle$ to $\langle x_2, \bar{x}_2, |s_1| + |s_1| \cdot |s_2|\rangle$, and so on until we label all arcs and all nodes of the policy tree. A path p from the root node to the last arc can be represented by the sequence of the node and arc labelings, i.e. $p = [\langle x_1, \bar{x}_1, 0\rangle, \langle s_1, \bar{s}_1\rangle, \ldots, \langle x_m, \bar{x}_m, m-1\rangle, \langle s_m, \bar{s}_m\rangle]$. Let Ψ denote the set of all distinct paths of a policy tree. For each $p \in \Psi$, we denote by $arcs(p)$ the sequence of all the arc labelings in p whereas $nodes(p)$ denotes the sequence of all node labelings in p. That is $arcs(p) = [\langle s_1, \bar{s}_1\rangle, \ldots, \langle s_m, \bar{s}_m\rangle]$ whereas $nodes(p) = [\langle x_1, \bar{x}_1, 0\rangle, \ldots, \langle x_m, \bar{x}_m, m-1\rangle]$. We denote by $\Omega = \{arcs(p) | p \in \Psi\}$ the set of all scenarios of the policy tree. The probability of $\omega \in \Omega$ is given by $\Pr\{\omega\} = \prod_{i=1}^{m} \Pr\{s_i = \bar{s}_i\}$, where $\Pr\{s_i = \bar{s}_i\}$ is the probability that stochastic variable s_i takes value $\bar{s}_i$.

Now consider a chance constraint $h \in C$ with a specified threshold level θ_h. Consider a policy tree $\mathcal{T}$ for the SCSP and a path $p \in \mathcal{T}$. Let $h_{\downarrow p}$ be the deterministic constraint obtained by substituting the stochastic variables in h with the corresponding values $(\bar{s}_i)$ assigned to these stochastic variables in $arcs(p)$. Let $\bar{h}_{\downarrow p}$ be the resulting tuple obtained by substituting the decision variables in $h_{\downarrow p}$ by the values $(\bar{x}_i)$ assigned to the corresponding decision variables in $nodes(p)$. We say that h is *satisfied with respect to a given policy tree* $\mathcal{T}$ iff

$$\sum_{p \in \Psi:\, \bar{h}_{\downarrow p} \in h_{\downarrow p}} \Pr\big\{arcs(p)\big\} \geqslant \theta_h.$$

Definition 1. Given an m-stage SCSP $\mathcal{P}$ and a policy tree $\mathcal{T}$, $\mathcal{T}$ is a satisfying policy tree to $\mathcal{P}$ iff every chance constraint of $\mathcal{P}$ is satisfied with respect to $\mathcal{T}$.

Example 1. Let us consider a two-stage SCSP in which $V_1 = \{x_1\}$ and $S_1 = \{s_1\}$, $V_2 = \{x_2\}$ and $S_2 = \{s_2\}$. Stochastic variable s_1 may take two possible values, 5 and 4, each with probability 0.5; stochastic variable s_2 may also take two possible values, 3 and 4, each with probability 0.5. The domain of x_1 is $\{1, \ldots, 4\}$, the domain of x_2 is $\{3, \ldots, 6\}$. There are two chance constraints[2] in C, c_1: $\Pr\{s_1 x_1 + s_2 x_2 \geqslant 30\} \geqslant 0.75$ and c_2: $\Pr\{s_2 x_1 = 12\} \geqslant 0.5$. In this case, the decision variable x_1 must be set to a unique value before random variables are observed, while decision variable x_2 takes a value that depends on the observed value of the random variable s_1. A possible solution to this SCSP is the satisfying policy tree shown in Fig. 1 in which $x_1 = 3$, $x_2^1 = 4$ and $x_2^2 = 6$, where x_2^1 is the value assigned to decision variable x_2, if random variable s_1 takes value 5, and x_2^2 is the value assigned to decision variable x_2, if random variable s_1 takes value 4. The four labeled paths of the above policy tree are as follows:

$$p_1 = \big[\langle x_1, 3, 0\rangle, \langle s_1, 5\rangle, \langle x_2, 4, 1\rangle, \langle s_2, 4\rangle\big],$$
$$p_2 = \big[\langle x_1, 3, 0\rangle, \langle s_1, 5\rangle, \langle x_2, 4, 1\rangle, \langle s_2, 3\rangle\big],$$
$$p_3 = \big[\langle x_1, 3, 0\rangle, \langle s_1, 4\rangle, \langle x_2, 6, 2\rangle, \langle s_2, 4\rangle\big],$$
$$p_4 = \big[\langle x_1, 3, 0\rangle, \langle s_1, 4\rangle, \langle x_2, 6, 2\rangle, \langle s_2, 3\rangle\big].$$

As the example shows, a solution to an SCSP is not simply an assignment of the decision variables in V to values, but it is instead a satisfying policy tree.

[2] In what follows, for convenience, we will denote a chance constraint by using the notation "$\Pr\{\langle cons\rangle\} \geqslant \theta$", meaning that constraint $\langle cons\rangle$, constraining decision and random variables, should be satisfied with probability greater or equal to θ.

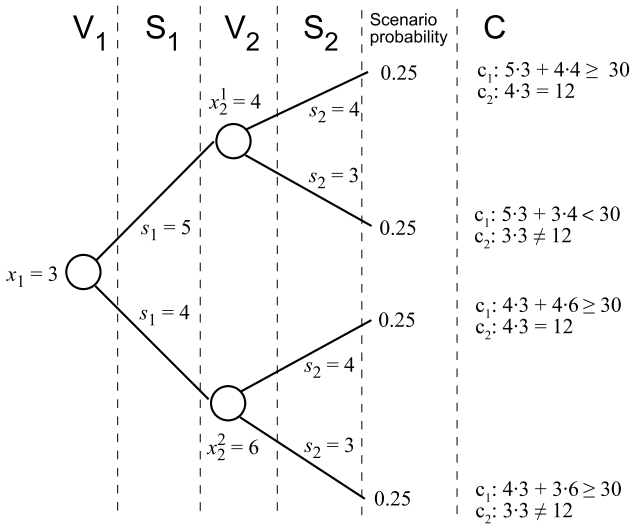

Fig. 1. Policy tree for the SCSP in Example 1.

Constraints:

(1) $(5x_1^1 + 4x_2^1 \geqslant 30) \leftrightarrow (Z_{c_1}^1 = 1)$ (6) $(4x_1^1 = 12) \leftrightarrow (Z_{c_2}^1 = 1)$

(2) $(5x_1^1 + 3x_2^1 \geqslant 30) \leftrightarrow (Z_{c_1}^2 = 1)$ (7) $(3x_1^1 = 12) \leftrightarrow (Z_{c_2}^2 = 1)$

(3) $(4x_1^1 + 4x_2^2 \geqslant 30) \leftrightarrow (Z_{c_1}^3 = 1)$ (8) $(4x_1^1 = 12) \leftrightarrow (Z_{c_2}^3 = 1)$

(4) $(4x_1^1 + 3x_2^2 \geqslant 30) \leftrightarrow (Z_{c_1}^4 = 1)$ (9) $(3x_1^1 = 12) \leftrightarrow (Z_{c_2}^4 = 1)$

(5) $\sum_{\omega=1}^{4} 0.25 Z_{c_1}^{\omega} \geqslant \theta_{c_1}$ (10) $\sum_{\omega=1}^{4} 0.25 Z_{c_2}^{\omega} \geqslant \theta_{c_2}$

Decision variables:

$x_1 \in \{1, 2, 3, 4\}$, $x_2^1 \in \{3, 4, 5, 6\}$,

$x_2^2 \in \{3, 4, 5, 6\}$, $Z_h^{\omega} \in \{0, 1\}\ \forall \omega = 1, \ldots, 4;\ \forall h \in \{c_1, c_2\}$.

Fig. 2. Deterministic equivalent CSP for Example 1.

4. Scenario-based approach to solve SCSPs

In [26], the authors discuss an equivalent scenario-based reformulation for SCSPs, which we shall call SBA in what follows. This reformulation makes it possible to compile SCSPs down into conventional (non-stochastic) CSPs. For example, the multi-stage SCSP described in Example 1 is compiled down to its deterministic equivalent CSP shown in Fig. 2. The decision variables x_1^1, x_2^1, and x_2^2 represent the nodes of the policy tree. The variable x_1 is decided at stage 1 so we have one copy of it (x_1^1) whereas since x_2 is to be decided at stage 2 and since s_1 has two values, we need two copies for x_2, namely x_2^1 and x_2^2. Chance constraint c_1 is compiled down into constraints $(1), \ldots, (5)$, whilst chance constraint c_2 is compiled down into constraints $(6), \ldots, (10)$. Constraints $(1), \ldots, (4)$ are reification constraints in which every binary decision variable $Z_{c_1}^{\omega}$ is 1 iff in scenario $\omega \in \{1, \ldots, 4\}$ constraint $\bar{s}_1 x_1^1 + \bar{s}_2 x_2^i \geqslant 30$ — where $i \in \{1, 2\}$ identifies the copy of decision variable x_2 associated with scenario ω — is satisfied. Finally, constraint (5) enforces that the satisfaction probability achieved must be greater than or equal to the required threshold $\theta_{c_1} = 0.75$. Similar reasoning applies to constraints $(6), \ldots, (10)$.

The scenario-based reformulation approach allows us to exploit the full power of existing constraint solvers. However, it has a number of serious drawbacks that might prevent it from being applied in practice.

Increased space requirements: For each chance constraint, $|\Omega|$ extra Boolean variables and at least $|\Omega| + 1$ extra constraints are introduced. This requires more space and might increase the solution time;

Weakened constraint propagation: The scenario-based reformulation heavily depends on reification constraints for constraint propagation. For this reason, it propagates weakly. Also, if the chance constraint involves a global constraint (e.g., $\Pr\{\text{alldiff}(x_1, s_1, x_2)\} \geqslant \theta$), then the corresponding reification constraints (e.g., $\text{alldiff}(x_1^1, \bar{s}_1, x_2^1) \leftrightarrow Z_w$) cannot simply be supported in an effective way in terms of propagation by most of the current constraint solvers. While a solver like Minion [10] effectively supports the above construct, in several other solvers it is often possible to

decompose the alldiff into a clique of binary not-equals constraints and pose instead reification constraints of the following form $(x_1^1 \neq \bar{s}_1 \wedge x_1^1 \neq x_2^1 \wedge x_2^1 \neq \bar{s}_1) \leftrightarrow Z_w$, but this is, of course, not an ideal solution.

5. Formal background

Like the approach in [28], in order to solve an m-stage SCSP, we introduce a decision variable for each node of the policy tree. Given an SCSP $\langle V, S, D, P, C, \theta, L\rangle$, we let $\mathcal{PT}$ be an array of decision variables indexed from 0 to $\mathcal{N}-1$ representing the space of all possible policy trees. The domains of these variables are defined as follows:

- $D(\mathcal{PT}[i]) = D(x_1)$, $i \in M_1 = \{0\}$,
- $D(\mathcal{PT}[i]) = D(x_2)$, $i \in M_2 = \{1, \ldots, |s_1|\}$,
- $D(\mathcal{PT}[i]) = D(x_3)$, $i \in M_3 = \{(1 + |s_1|), \ldots, (|s_1| \cdot |s_2| + |s_1|)\}$,
- ...
- $D(\mathcal{PT}[i]) = D(x_m)$, $i \in M_m = \{(1 + |s_1| \cdot |s_2| \cdot \ldots \cdot |s_{m-2}|), \ldots, (|s_1| \cdot |s_2| \cdot \ldots \cdot |s_{m-1}| + |s_1| \cdot |s_2| \cdot \ldots \cdot |s_{m-2}|)\}$,

where M_j represents the set of indexes — in our node labeling — that appear at depth j in the policy tree, $j = \{1, \ldots, m\}$. This array of decision variables is shared among the constraints in the model similarly to what happens with decision variables in classic CSPs.

Definition 2. Given a chance constraint $h \in C$ and a policy tree decision variable array $\mathcal{PT}$, a value v in the domain of $\mathcal{PT}[i]$ is **consistent with respect to** h iff there exists an assignment of values to variables in $\mathcal{PT}$ that is a satisfying policy with respect to h, in which $\mathcal{PT}[i] = v$.

Definition 3. A chance constraint $h \in C$ is **generalized arc-consistent** iff every value in the domain of every variable in $\mathcal{PT}$ is consistent with respect to h.

Definition 4. An SCSP is **generalized arc-consistent** iff every chance constraint is generalized arc-consistent.

Maintaining GAC on an SCSP is NP-hard in general as solving an SCSP is PSPACE in general. In what follows, we show that maintaining GAC on a global chance constraint can be intractable even when maintaining GAC on the corresponding deterministic version of that constraint is tractable. In particular, we show that maintaining GAC on the alldiff global chance constraint is NP-hard while maintaining GAC on the deterministic alldiff constraint is polynomial [20].

We show a reduction from the problem *TwoAllDiff* of finding a solution to two alldiff constraints which is NP-hard [14] to an SCSP:

$$\text{alldiff}(x_1, \ldots, x_i, \ldots, x_n) \wedge \text{alldiff}(x_i, \ldots, x_n, \ldots, x_m).$$

Assume without loss of generality that $i - 1 = m - n$. We can always add dummy variables to the alldiff constraint with less number of variables.

Given an instance for the *TwoAllDiff* problem, we construct an instance of a two-stage SCSP as follows. We introduce $n + m$ decision variables and one stochastic variable r whose domain is composed of only two values and the probability of each is $\frac{1}{2}$. The decision variables are divided into two groups. In the first group, which is decided at the first stage, we introduce a decision variable z_j whose domain is the same as x_j for all $j \in \{i, \ldots, n\}$. In the second group, we introduce $i - 1$ second stage y_j variables where the domain of y_j is the union of the domain of x_j and x_{j+n} for all $j \in \{1, \ldots, i-1\}$. We introduce three chance constraints. The first chance constraint is the global alldiff chance constraint which constrains all the decision variables. The other two chance constraints restrict the domains of the second stage variables as follows. If we are in the first scenario, the second stage variable's domains are restricted to take the same domain as the non-overlapping variables in the first alldiff constraint whereas if we are in the second scenario, the second stage variable domains are restricted to take the same domain as the non-overlapping variables in the second alldiff constraint. Fig. 3 shows the complete SCSP.

Theorem 1. *TwoAllDiff has a solution iff the two stage SCSP in Fig. 3 has a satisfying policy tree.*

Proof *(sketch).* The deterministic equivalent CSP for the two stage SCSP in Fig. 3 generated using a scenario-based approach is indeed equivalent to the *TwoAllDiff* up to variable renaming and is shown in Fig. 4. Indeed, the $z_i, \ldots, z_n$ correspond to $x_i, \ldots, x_n$, respectively. The $y_1^1, \ldots, y_{i-1}^1$ correspond to $x_1, \ldots, x_{i-1}$, respectively. Finally, $y_1^2, \ldots, y_{i-1}^2$ correspond to $x_{n+1}, \ldots, x_m$, respectively. Note that the unary constraints in the two-stage SCSP are absorbed into the domains. □

A satisfying policy tree of the two-stage SCSP in Fig. 3 corresponds to a solution to *TwoAllDiff* as follows. The assignment to the first stage variables $z_i, \ldots, z_n$ correspond to an assignment to $x_i, \ldots, x_n$, respectively. The assignment of the second

Constraints:
alldiff$(z_i, \ldots, z_n, y_1, \ldots, y_{i-1})$
$r_1 = 1 \rightarrow y_j \in D(x_j) \quad \forall j \in 1, \ldots, i-1$
$r_1 = 2 \rightarrow y_j \in D(x_{j+n}) \quad \forall j \in 1, \ldots, i-1$

Decision variables:
$z_j \in D(x_j) \quad \forall j \in i, \ldots, n$
$y_j \in D(x_j) \cup D(x_{j+n}) \quad \forall j \in 1, \ldots, i-1$

Random variables:
$r_1 \in \{1, 2\}$

Stage structure:
$V_1 = \{z_i, z_2, \ldots, z_n\} \quad V_2 = \{y_1, y_2, \ldots, y_{i-1}\}$
$S_1 = \{r_1\} \quad S_2 = \{\}$
$L = [\langle V_1, S_1\rangle, \langle V_2, S_2\rangle]$

Fig. 3. A two-stage SCSP.

Constraints:
(1) alldiff$(z_i, \ldots, z_n, y^1_1, \ldots, y^1_{i-1})$
(2) alldiff$(z_i, \ldots, z_n, y^2_1, \ldots, y^2_{i-1})$

Decision variables:
$z_j \in D(x_j) \quad \forall j \in \{i, \ldots, n\}$
$y^1_j \in D(x_j) \quad \forall j \in \{1, \ldots, i-1\}$
$y^2_j \in D(x_{j+n}) \quad \forall j \in \{1, \ldots, i-1\}$

Fig. 4. Deterministic equivalent CSP of the SCSP of Fig. 3.

stage variables under the first scenario ($r = 1$) correspond to an assignment to the remaining variables of the first alldiff constraint whereas the assignment of the second stage variables under the second scenario ($r = 2$) correspond to an assignment to the remaining variables of the second alldiff constraint. Since finding a satisfying policy tree to the SCSP in Fig. 3 is NP-hard, then achieving GAC is also NP-hard [4].

Finally, it is straightforward to prove that GAC on the SCSP in Fig. 3 is equivalent to GAC on the alldiff global chance constraint as the other two chance constraints only restrict the domains on the second stage variables in the policy tree. Therefore, maintaining GAC on the alldiff global chance constraint is NP-hard as well.

We can easily also show that maintaining bounds consistency (BC), a weaker consistency than GAC, on the alldiff global chance constraint is NP-hard. We consider the *ThreeAllDiff* problem in which we want to achieve BC. It is shown in [9] that this problem is intractable when we have more than two overlapping alldiff constraints. We can easily generalize the previous reduction in a straightforward manner to also show that maintaining BC on the alldiff global chance constraint is NP-hard. Any instance of the *ThreeAllDiff* problem in which we want to achieve BC can be transformed into an instance of a two-stage SCP very similar to the one in Fig. 3. We, however, need to have three values in the domain of the stochastic variable instead of two, each with probability $\frac{1}{3}$. Furthermore, the first stage decision variables will correspond to the overlapping variables in the *ThreeAllDiff* like in the previous reduction. The second stage decision variables will be used to represent the non-overlapping variables in the same way as we did in the previous reduction where each scenario will correspond to one alldiff constraint.

For convenience, given a chance constraint $h \in C$, we redefine $h_{\downarrow p}$ as the deterministic constraint obtained by substituting every decision variable x_i in h with decision variable $\mathcal{PT}[k]$ — where $\langle x_i, -, k\rangle$ is an element in $nodes(p)$ — and every stochastic variable s_i with the corresponding value $(\bar{s}_i)$ assigned to s_i in $arcs(p)$. Note that the deterministic constraint $h_{\downarrow p}$ is a classical constraint, so a value v in the domain of any decision variable *is consistent* iff there exist compatible values for all other variables such that $h_{\downarrow p}$ is satisfied, otherwise v is inconsistent. Denote by $h^{i,v}_{\downarrow p}$ constraint $h_{\downarrow p}$ in which decision variable $\mathcal{PT}[i]$ is set to v. $h^{i,v}_{\downarrow p}$ *is consistent* if value v in $D(\mathcal{PT}[i])$ is consistent w.r.t. $h_{\downarrow p}$.

Example 1a. Let h be chance constraint c_1 and p be path p_1 in Example 1. Let $i = 2$; according to our labeling $\mathcal{PT}[2] = x_2$ and $D(\mathcal{PT}[2]) = \{3, \ldots, 6\}$. Let $v = 4$; then from the solution previously presented it is clear that $h^{i,v}_{\downarrow p}$ *is consistent* since value 4 in $D(\mathcal{PT}[2])$ is consistent w.r.t. $h_{\downarrow p}$.

Let $\Psi_{i,h} = \{p \in \Psi | h_{\downarrow p}$ *constrains* $\mathcal{PT}[i]\}$. We introduce $f[i, v, h]$ as follows:

$$f[i, v, h] = \sum_{p \in \Psi_i:\ h^{i,v}_{\downarrow p} \text{ is consistent}} \Pr\big(arcs(p)\big),$$

Algorithm 1: Generic non-incremental filtering algorithm.

```
   input : h; PT; A.
   output: Filtered PT with respect to h.
 1 begin
 2     for each i ∈ {0, ..., N − 1} do
 3         for each v ∈ D(PT[i]) do
 4             f[i, v] ← 0;
 5     for each p ∈ Ψ do
 6         Create a copy c of h_↓p and of the decision variables it constrains;
 7         Enforce GAC on c using A;
 8         for each index i of the variables in c do
 9             for each v in domain of the copy of PT[i] do
10                 f[i, v] ← f[i, v] + Pr{arcs(p)};
11         delete c and the respective copies of the decision variables;
12     for each i ∈ {0, ..., N − 1} do
13         max[i] ← 0;
14         for each v ∈ D(PT[i]) do
15             max[i] ← max(max[i], f[i, v]);
16     for each k ∈ {1, ..., m} do
17         g[k] ← 0;
18         for each i ∈ M_k do
19             g[k] ← g[k] + max[i];
20     for each k ∈ {1, ..., m} do
21         for each i ∈ M_k do
22             for each v ∈ PT[i] do
23                 if g[k] − max[i] + f[i, v] < θ_h then
24                     prune value v from D(PT[i]);
25 end
```

where $f[i, v, h]$ is the sum of the probabilities of the scenarios in which value v in the domain of $\mathcal{PT}[i]$ is consistent. To keep notation as compact as possible, since we always refer to a "generic" constraint h, in what follows we write Ψ_i in place of $\Psi_{i,h}$ and $f[i, v]$ in place of $f[i, v, h]$. As the next proposition shows, one can exploit $f[i, v]$ to identify a subset of the inconsistent values.

Proposition 1. *For any $i \in M_k$ and value $v \in D(\mathcal{PT}[i])$, if*

$$f[i, v] + \sum_{j \in M_k, j \neq i} \max(j) < \theta_h,$$

then v is inconsistent with respect to h; where $\max(j) = \max\{f[j, v] \mid v \in D(\mathcal{PT}[j])\}$.

Proof *(sketch).* The assignment $\mathcal{PT}[i] = v$ is consistent w.r.t. h iff the satisfaction probability of h is greater or equal to θ_h. From the definition of $f[i, v]$ and of $\max(j)$ it follows that if $f[i, v] + \sum_{j \in M_k, j \neq i} \max(j) < \theta_h$, when $\mathcal{PT}[i] = v$, the satisfaction probability of h is less than θ_h even if we choose the best possible value for all the other variables in M_k. □

6. Generic filtering algorithms

We now describe our generic filtering strategy for chance constraints. We distinguish between two cases: the case when $\theta_h < 1$ and the case where $\theta_h = 1$. In the first case, we design a specialized filtering algorithm whereas for the second case we provide a reformulation approach that is more efficient. Both methods, however, are parameterized with a filtering algorithm $\mathcal{A}$ for the deterministic constraints $h_{\downarrow p}$ for all $p \in \Psi$ that maintains GAC (or any other level of consistency). This allows us to reuse existing filtering algorithms in current constraint solvers and makes our filtering algorithms generic and suitable for any global chance constraint.

6.1. Case 1 ($\theta_h < 1$)

Algorithm 1 takes as input chance constraint h, $\mathcal{PT}$, and a propagator $\mathcal{A}$. It filters from $\mathcal{PT}$ inconsistent values with respect to h. For each decision variable and each value in its domain, we initialize $f[i, v]$ to 0 (line 2). At line 5, we iterate through the scenarios in Ψ. For each scenario, we create a copy c of constraint $h_{\downarrow p}$ and of the decision variables

Table 1
Example of inconsistent values gone undetected in Example 2.

$\mathcal{PT}[0]$	$f[0,v]$	$\mathcal{PT}[1]$	$f[1,v]$	$\mathcal{PT}[2]$	$f[2,v]$
$\underline{1}$	0.75	$\underline{1}$	0.25	$\underline{3}$	0.5
2	0.25	$\underline{2}$	0.25		

it constrains. Then we enforce GAC on c using $\mathcal{A}$. For each i such that $\mathcal{PT}[i]$ is constrained by $h_{\downarrow p}$, we iterate through the domain of the copy of the decision variable and, if a given value v has support, we add the probability associated with the current scenario to the respective $f[i,v]$ (line 10). It should be noted that, for each scenario, constraint c is dynamically generated every time the filtering algorithm runs, and also that these constraints are never posted into the model. They are only used to reduce the domains of the copies of the associated decision variables. At line 12, for each variable $i \in \{0,\ldots,\mathcal{N}-1\}$ we compute the maximum support probability $f[i,v]$ provided by a value v in the domain of $\mathcal{PT}[i]$, and we store it at max[i]. At line 16, for each stage $k \in \{1,\ldots,m\}$, we store in $g[k]$ the sum of the max[i] of all variables $i \in M_k$. Finally (line 20), at stage k we prune from $D(\mathcal{PT}[i])$ any value v that makes $g[k]$ strictly smaller than θ_h when we replace max[i] in $g[k]$ with $f[i,v]$.

Theorem 2. *Algorithm 1 is a sound filtering algorithm.*

Proof *(sketch).* **Soundness.** When a value v is pruned by Algorithm 1 at line 24, $g[k]$ becomes strictly smaller than θ_h when we replace max[i] in $g[k]$ with $f[i,v]$. Indeed $g[k]$ is thus equal to $f[i,v] + \sum_{j \in M_k, j \neq i} \max(j)$ which makes Proposition 1 true. Thus, any pruned value v is inconsistent. □

Algorithm 1 fails to prune some inconsistent values because such values are supported by values that may become inconsistent at a later stage of the algorithm. We illustrate these situations with an example.

Example 2. Consider a 2-stage SCSP in which $V_1 = \{x_1\}$, where $x_1 \in \{1,2\}$, $S_1 = \{s_1\}$, where $s_1 \in \{a,b\}$, $V_2 = \{x_2\}$, where $x_2 \in \{1,2,3\}$, and $S_2 = \{s_2\}$, where $s_2 \in \{a,b\}$. Let $\Pr\{s_i = j\} = 0.5$ for all $i \in \{1,2\}$ and $j \in \{a,b\}$. Let $h(x_1,x_2,s_1,s_2)$ be the chance constraint with $\theta_h = 0.75$. In this constraint, for the first scenario ($s_1 = a$ and $s_2 = a$) the only consistent values for $\mathcal{PT}[0]$ and $\mathcal{PT}[1]$ are 1 and 2 respectively. For the second scenario ($s_1 = a$ and $s_2 = b$) the only consistent values for $\mathcal{PT}[0]$ and $\mathcal{PT}[1]$ are 2 and 1 respectively. For the third scenario ($s_1 = b$ and $s_2 = a$) the only consistent values for $\mathcal{PT}[0]$ and $\mathcal{PT}[2]$ are 1 and 3 respectively. For the fourth scenario ($s_1 = b$ and $s_2 = b$) the only consistent values for $\mathcal{PT}[0]$ and $\mathcal{PT}[2]$ are 1 and 3 respectively. That is, the set of allowed tuples for the deterministic version of h is $\{\langle 1,2,a,a\rangle, \langle 2,1,b,a\rangle, \langle 1,3,b,a\rangle, \langle 1,3,b,b\rangle\}$. Our algorithm originally introduces three decision variables $\mathcal{PT}[0] \in \{1,2\}$, $\mathcal{PT}[1] \in \{1,2,3\}$, and $\mathcal{PT}[2] \in \{1,2,3\}$. Assume that at some stage during search, the domains become $\mathcal{PT}[0] \in \{1,2\}$, $\mathcal{PT}[1] \in \{1,2\}$, and $\mathcal{PT}[2] \in \{3\}$. In Table 1, the values that are not pruned by Algorithm 1 when $\theta = 0.75$ are underlined. Only value 2 in the domain of $\mathcal{PT}[0]$ is pruned. But value 2 was providing support to value 1 in the domain of $\mathcal{PT}[1]$. This goes undetected by the algorithm and value 1 for $\mathcal{PT}[1]$ no longer provides $f[1,1] = 0.25$ satisfaction, but 0. Thus, there exists no satisfying policy in which $\mathcal{PT}[1] = 1$.

We can easily remedy this problem by repeatedly calling Algorithm 1 until we reach a fixed-point and no further pruning is done. For the rest of the paper, we refer to this modified algorithm as Algorithm 1 as well.

Theorem 3. *Algorithm 1 runs in $O(|\Omega| \cdot a \cdot \mathcal{N}^2 \cdot d^2)$ time and in $O(\mathcal{N} \cdot d + p)$ space where a is the time complexity of $\mathcal{A}$, p is its space complexity, and d is the maximum domain size.*

Proof *(sketch).* **Time complexity.** In the worst case, Algorithm 1 needs to be called $\mathcal{N} \cdot d$ times in order to prune at each iteration just one inconsistent value. At each of these iterations, the time complexity is dominated by complexity of line 7 assuming that $|\Omega| \gg |V|$. Enforcing GAC on each of the $|\Omega|$ constraints runs in a time using algorithm $\mathcal{A}$. In the worst case, we need to repeat this whole process $\mathcal{N} \cdot d$ times in order to prune at each iteration just one inconsistent value. Thus the time complexity of this step is in $|\Omega| \cdot a \cdot \mathcal{N} \cdot d$. The overall time complexity is therefore in $O(|\Omega| \cdot a \cdot \mathcal{N}^2 \cdot d^2)$ time.

Space complexity. The space complexity is dominated by the size of $\mathcal{PT}$ and by the space complexity of $\mathcal{A}$. $\mathcal{PT}$ requires $\mathcal{N} \cdot d$ space whereas $\mathcal{A}$ requires p space. Therefore, the modified algorithm runs in $O(\mathcal{N} \cdot d + p)$ space. □

In Table 2 we report the pruned values for Example 1 achieved by Algorithm 1. The values that are not pruned are underlined. Note that if we propagate the constraints in the model generated according to the approach described in [26] and shown in Fig. 2, no value is pruned at all.

Even though Algorithm 1 is a sound filtering algorithm, it is unfortunately still incomplete as maintaining GAC on h is intractable in general.

Table 2
Pruning for Example 1 after calling Algorithm 1.

$\mathcal{PT}[0]$	$f[0,v]$	$\mathcal{PT}[1]$	$f[1,v]$	$\mathcal{PT}[2]$	$f[2,v]$
1	0.0	$\underline{3}$	0.25	3	0.0
2	0.5	$\underline{4}$	0.5	$\underline{4}$	0.25
$\underline{3}$	1.0	$\underline{5}$	0.5	$\underline{5}$	0.5
$\underline{4}$	1.0	$\underline{6}$	0.5	$\underline{6}$	0.5

Table 3
Filtered domains in Example 3.

$\mathcal{PT}[0]$	$f[0,v]$	$\mathcal{PT}[1]$	$f[1,v]$	$\mathcal{PT}[2]$	$f[2,v]$
$\underline{1}$	1	$\underline{1}$	0.5	$\underline{1}$	0.5
$\underline{2}$	1	$\underline{2}$	0.5	$\underline{2}$	0.5

Theorem 4. *The level of consistency achieved by Algorithm 1 on global chance constraint h is weaker than GAC on h.*

Proof *(Example 3).* Consider a 2-stage SCSP where $V_1 = \{x_1\}$ where $x_1 \in \{1,2\}$, $S_1 = \{s_1\}$ where $s_1 \in \{a,b\}$, $V_2 = \{x_2\}$ where $x_2 \in \{1,2\}$, and $S_2 = \{s_2\}$ where $s_2 \in \{a,b\}$. Let $Pr\{s_i = j\} = 0.5$ for all $i \in \{1,2\}$ and $j \in \{a,b\}$. Let $h(x_1,x_2,s_1,s_2)$ be the chance constraint with $\theta_h = 0.75$. Furthermore, for the first scenario ($s_1 = a$ and $s_2 = a$) the consistent tuples for x_1 and x_2 are in $\{\langle 1,1\rangle\langle 2,1\rangle\langle 2,2\rangle\}$. For the second scenario ($s_1 = a$ and $s_2 = b$) the consistent tuples for x_1 and x_2 are in $\{\langle 1,2\rangle\langle 2,1\rangle\langle 2,2\rangle\}$. For the third scenario ($s_1 = b$ and $s_2 = a$) the consistent tuples for x_1 and x_2 are in $\{\langle 1,1\rangle\langle 2,1\rangle\langle 2,2\rangle\}$. For the fourth scenario ($s_1 = b$ and $s_2 = b$) the consistent tuples for x_1 and x_2 are in $\{\langle 1,2\rangle\langle 2,1\rangle\langle 2,2\rangle\}$. That is the set of allowed tuples for the deterministic equivalent constraint of h is

$$\big\{\langle 1,1,a,a\rangle, \langle 2,1,a,a\rangle, \langle 2,2,a,a\rangle, \langle 1,2,a,b\rangle, \langle 2,1,a,b\rangle, \langle 2,2,a,b\rangle,$$
$$\langle 1,1,b,a\rangle, \langle 2,1,b,a\rangle, \langle 2,2,b,a\rangle, \langle 1,2,b,b\rangle, \langle 2,1,b,b\rangle, \langle 2,2,b,b\rangle\big\}.$$

Algorithm 1 introduces three decision variables $\mathcal{PT}[i] \in \{1,2\}$ for all $i \in \{0,1,2\}$. Table 3 shows the result of Algorithm 1. None of the values is pruned, but there exists no satisfying policy in which $\mathcal{PT}[0] = 1$. □

6.2. Case 2 ($\theta_h = 1$)

When $\theta_h = 1$ the global chance constraint h can be reformulated as

$$h_{\downarrow p}, \quad \forall p \in \Psi.$$

If all deterministic constraints are simultaneously GAC, then this reformulation is equivalent to Algorithm 1. Nevertheless, even in this special case, we still lose in terms of pruning.

Theorem 5. *GAC on h is stronger than GAC on the reformulation.*

Proof *(sketch).* We consider the same example as in the previous proof but with $\theta_h = 1$ instead. All deterministic constraints are simultaneously GAC, but $\mathcal{PT}[i] = 1$ cannot be extended to any satisfying policy. □

7. Incremental filtering algorithm

Filtering Algorithm 1 can be made incremental by introducing backtrackable objects that keep track of the scenarios for which a domain wipeout has already been detected or in which a given value has been already pruned at some earlier branching point during search. It is clear that tracking information at scenario level leads to a "lightweight" incremental algorithm while tracking information at value level leads to a more "memory intensive" algorithm. In Section 7.1 we introduce a lightweight incremental extension to Algorithm 1. A memory-intensive one is introduced in Section 7.2.

7.1. Lightweight incremental algorithm

Let $\mathcal{BS}$ denote a *stored bit set* of size $|\Omega|$. A *stored bit set* is an array of bits that is automatically restored to its previous state at each backtrack during search. Each bit $\mathcal{BS}[p]$ is uniquely associated with path $p \in \Psi$ and it can be either set to 1 or 0. $\mathcal{BS}$ is created when the global chance constraint is initialized. Upon creation, every bit in $\mathcal{BS}$ is set to 1. Let

$$\Pr\{\mathcal{BS}\} = \sum_{p \in \Psi:\ \mathcal{BS}[p]=1} \Pr\big\{arcs(p)\big\}.$$

Algorithm 2: Generic lightweight incremental filtering algorithm.

input : h; $\mathcal{PT}$; $\mathcal{A}$; $\mathcal{BS}$.
output: Filtered $\mathcal{PT}$ with respect to h.

```
1  begin
2    for each i ∈ {0, ..., 𝒩 − 1} do
3      for each v ∈ D(𝒫𝒯[i]) do
4        f[i, v] ← 0;
5        d[i, v] ← 0;
6    for each p ∈ Ψ do
7      if ℬ𝒮[p] = 1 then
8        Create a copy c of h↓p and of the decision variables it constrains;
9        Enforce GAC on c using 𝒜;
10       if c is inconsistent then
11         ℬ𝒮[p] = 0;
12         if Pr{ℬ𝒮} < θh then
13           backtrack;
14       else
15         for each index i of the variables in c do
16           for each v in domain of the copy of 𝒫𝒯[i] do
17             f[i, v] ← f[i, v] + Pr{arcs(p)};
18           for each v pruned from domain of the copy of 𝒫𝒯[i] do
19             d[i, v] ← d[i, v] + Pr{arcs(p)};
20             if Pr{ℬ𝒮} − d[i, v] < θh then
21               prune value v from D(𝒫𝒯[i]);
22         delete c and the respective copies of the decision variables;
23   for each i ∈ {0, ..., 𝒩 − 1} do
24     max[i] ← 0;
25     for each v ∈ D(𝒫𝒯[i]) do
26       max[i] ← max(max[i], f[i, v]);
27   for each k ∈ {1, ..., m} do
28     g[k] ← 0;
29     for each i ∈ Mk do
30       g[k] ← g[k] + max[i];
31   for each k ∈ {1, ..., m} do
32     for each i ∈ Mk do
33       for each v ∈ 𝒫𝒯[i] do
34         if g[k] − max[i] + f[i, v] < θh then
35           prune value v from D(𝒫𝒯[i]);
36 end
```

As in the previous case, our algorithm is parameterized with a filtering algorithm $\mathcal{A}$ for the deterministic constraint $h_{\downarrow p}$.

Algorithm 2 takes as input chance constraint h, $\mathcal{PT}$, a propagator $\mathcal{A}$, and a stored bit set $\mathcal{BS}$. It filters from $\mathcal{PT}$ inconsistent values with respect to h. For each decision variable and each value in its domain, we initialize $f[i, v]$ and $d[i, v]$ to 0 (line 2); $d[i, v]$ is an auxiliary accumulator that tracks the total probability associated with scenarios in which value v in $D(\mathcal{PT}[i])$ does not have support. At line 6, we iterate through the scenarios in Ψ. For each scenario $p \in \Psi$, at line 7 if $h_{\downarrow p}$ has been already detected to be inconsistent at earlier branches in the search tree ($\mathcal{BS}[p] = 0$) we do nothing, otherwise ($\mathcal{BS}[p] = 1$) we create a copy c of constraint $h_{\downarrow p}$ and of the decision variables it constrains. Then we enforce GAC on c using $\mathcal{A}$ (line 9). Recall that $h_{\downarrow p}$ is a classical constraint, so we can enforce consistency by using standard propagation techniques. At line 10, if GAC produces a domain wipeout in c at line 9, we set the bit $\mathcal{BS}[p]$ to 0 and, at line 12, we check if the remaining scenarios can provide an overall probability that exceeds θ_h. If they cannot, we backtrack. On the other hand, if GAC does not produce a domain wipeout in c at line 9, at line 14 for each i such that $\mathcal{PT}[i]$ is constrained by $h_{\downarrow p}$, we iterate through the domain of the copy of the decision variable and, if a given value v has support — i.e. it has not been pruned when we enforced GAC on c at line 9 — we add the probability associated with the current scenario to the respective $f[i, v]$ (line 17); conversely, if a value v does not have support — i.e. it has been pruned when we enforced GAC on c at line 9 — we add the probability associated with the current scenario to the respective $d[i, v]$ (line 19) and we immediately check if it is possible to prune such a value (line 20). The remaining lines of the algorithm are identical to those described in Algorithm 1.

To summarize, in Algorithm 2 backtrack may occur at line 13, informally speaking if in "too many" scenarios we have observed a domain wipeout — that is if we cannot possibly achieve the prescribed satisfaction probability with the remain-

ing scenarios. Furthermore, backtrack may occur if we observe a domain wipeout after pruning value v from $D(\mathcal{PT}[i])$ at lines 21 and 35.

Theorem 6. *Algorithm 2 is a sound filtering algorithm.*

Proof *(sketch).* **Soundness.** A value is either pruned at line 35 or/and line 20 or backtracking occurs at line 12. The pruning that happens at line 35 is similar to the one that happens in the non-incremental algorithm. $g[k]$ becomes strictly smaller than θ_h when we replace $\max[i]$ in $g[k]$ with $f[i,v]$. Indeed $g[k]$ is thus equal to $f[i,v] + \sum_{j \in M_k, j \neq i} \max(j)$ which makes Proposition 1 true. The pruned value v is inconsistent. The eager pruning at line 20 is a weakened reformulation of the condition verified at line 35. When backtracking occurs at line 12, the probability associated with scenarios for which an inconsistency has been detected amounts to a value greater than $1 - \theta_h$. Therefore there exists no policy tree $\mathcal{T}$ for which

$$\sum_{p \in \Psi:\, \bar{h}_{\downarrow p} \in h_{\downarrow p}} \Pr\{arcs(p)\} \geqslant \theta_h. \qquad \square$$

There are two key differences between Algorithms 1 and 2: the eager pruning at line 20, which is a weakened reformulation of the condition verified at line 35; and the backtracking at line 12. As previously remarked, Algorithm 1 fails to prune some inconsistent values because such values are supported by values that may become inconsistent at a later stage of the algorithm. Eager pruning tries to partially overcome this issue by proactively removing inconsistent values. This, in turn, may affect the assessment carried out on subsequent scenarios. Eventually, eager pruning may reduce the number of calls required to reach a fixed point.

As in the previous case, we can call Algorithm 2 until we reach a fixed-point and no further pruning is done. We denote as Algorithm 2 this modified algorithm as well.

Theorem 7. *Algorithm 2 runs in $O(|\Omega| \cdot a \cdot \mathcal{N}^2 \cdot d^2)$ time and in $O(\mathcal{N} \cdot d + p + |\Omega|)$ space where a is the time complexity of $\mathcal{A}$, p is its space complexity, and d is the maximum domain size.*

Proof *(sketch).* **Time complexity.** The proof is identical to that provided for Algorithm 1.

Space complexity. The space complexity is now dominated by the size of $\mathcal{BS}$, of $\mathcal{PT}$ and by the space complexity of $\mathcal{A}$. $\mathcal{BS}$ has size $|\Omega|$, in fact we store one bit per scenario. $\mathcal{PT}$ requires $\mathcal{N} \cdot d$ space whereas $\mathcal{A}$ requires p space. Therefore, the algorithm runs in $O(\mathcal{N} \cdot d + p + |\Omega|)$ space. $\square$

7.2. Memory-intensive incremental algorithm

The approach discussed in Section 7.1 keeps track, during search, of the scenarios which have already generated a domain wipeout. In contrast to the naive filtering presented in Algorithm 1, this enhanced algorithm therefore avoids propagating again over a scenario $p \in \Psi$ for which $h_{\downarrow p}$ is disentailed. The filtering effectiveness, i.e. the number of values pruned from decision variable domains, is not affected, while the filtering efficiency is clearly improved since a number of unnecessary runs for algorithm $\mathcal{A}$ are avoided when possible. This approach has memory requirements that are comparable to those of Algorithm 1, in fact this enhanced algorithm simply requires a backtrackable stored bit set of size $|\Omega|$ in order to memorize which scenarios have already generated a domain wipeout. In this section, we introduce an alternative memory intensive strategy that keeps track during search of which values in decision variable domains have already been pruned for each possible scenario.

Similarly to the approach discussed in Section 7.1, we introduce a stored bit set $\mathcal{BS}$, in which each bit $\mathcal{BS}[p]$ is uniquely associated with path $p \in \Psi$ and it can be either set to 1 or 0. $\mathcal{BS}$ keeps track, during search, of which scenarios have already generated a domain wipeout. $\mathcal{BS}$ is created when the global chance constraint is initialized. Upon creation, every bit in $\mathcal{BS}$ is set to 1. We recall that

$$\Pr\{\mathcal{BS}\} = \sum_{p \in \Psi:\, \mathcal{BS}[p]=1} \Pr\{arcs(p)\}.$$

In contrast to the approach discussed in Section 7.1, in this case we also associate a stored bit set of size $|\Psi_i|$ with each value $v \in D(\mathcal{PT}[i])$, for $i = 0, \ldots, \mathcal{N} - 1$. Let us denote this backtrackable object as $\mathcal{VBS}[v, \mathcal{PT}[i], p]$, where $p \in \Psi_i$ denotes a scenario in which $h_{\downarrow p}$ constrains $\mathcal{PT}[i]$. For each scenario $p \in \Psi_i$, decision variable $\mathcal{PT}[i]$ and value v, $\mathcal{VBS}[v, \mathcal{PT}[i], p] = 1$ if the value has not been already pruned from $D(\mathcal{PT}[i])$ in scenario p, otherwise $\mathcal{VBS}[v, \mathcal{PT}[i], p] = 0$. Also $\mathcal{VBS}$ is created when the global chance constraint is initialized. Upon creation, every bit in $\mathcal{VBS}$ is set to 1.

It is clear that $f[i,v]$, which in the previous algorithms denoted a value recomputed from scratch at each propagation run, is now functionally dependent on $\mathcal{VBS}$ and $\mathcal{BS}$. We can therefore write

$$f[i,v] = \sum_{p \in \Psi_i} \Pr\{arcs(p)\} \cdot \mathcal{BS}[p] \cdot \mathcal{VBS}[v, \mathcal{PT}[i], p]. \tag{1}$$

Algorithm 3: Generic memory-intensive incremental filtering algorithm.

```
   input : h; PT; A; BS; VBS; PT[t].
   output: Filtered PT with respect to h.
 1 begin
 2   for each p ∈ Ψ_t do
 3     if BS[p] = 1 then
 4       Create a copy c of h_↓p and of the decision variables it constrains;
         exclude, for any given decision variable d in c every value v for which VBS[v, d, p] = 0;
 5       Enforce GAC on c using A;
 6       if c is inconsistent then
 7         BS[p] = 0;
 8         if Pr{BS} < θ_h then
 9           backtrack;
10       else
11         for each index i of the variables in c do
12           for each v pruned from domain of the copy of PT[i] do
13             VBS[v, PT[i], p] = 0;
14         delete c and the respective copies of the decision variables;
15   for each i ∈ {0, ..., N − 1} do
16     max[i] ← 0;
17     for each v ∈ D(PT[i]) do
18       max[i] ← max(max[i], f[i, v]);
19   for each k ∈ {1, ..., m} do
20     g[k] ← 0;
21     for each i ∈ M_k do
22       g[k] ← g[k] + max[i];
23   for each k ∈ {1, ..., m} do
24     for each i ∈ M_k do
25       for each v ∈ PT[i] do
26         if g[k] − max[i] + f[i, v] < θ_h then
27           prune value v from D(PT[i]);
28 end
```

This is simply the sum of the probabilities of those scenarios in which $\mathcal{PT}[i]$ is constrained by $h_{\downarrow p}$ and in which neither the whole constraint $h_{\downarrow p}$ is disentailed nor value v is inconsistent.

As in the previous cases, our algorithm is parameterized with a filtering algorithm $\mathcal{A}$ for the deterministic constraints $h_{\downarrow p}$. Due to the increasing granularity at which we track inconsistency, it is relevant in this case to identify which decision variable originated the event that triggered the current propagation run, let this decision variable be $\mathcal{PT}[t]$.

Algorithm 3 takes as input chance constraint h, $\mathcal{PT}$, a propagator $\mathcal{A}$, stored bit sets $\mathcal{BS}$ and $\mathcal{VBS}$, and the source of the propagation event, decision variable $\mathcal{PT}[t]$. It filters from $\mathcal{PT}$ inconsistent values with respect to h. At line 2, we iterate through the scenarios in Ψ_t. Recall that these are all the scenarios $p \in \Psi$ in which the decision variable $\mathcal{PT}[t]$ that triggered the propagation is constrained by $h_{\downarrow p}$. For each scenario $p \in \Psi_t$, at line 3, if $h_{\downarrow p}$ has been already detected to be inconsistent at earlier branches in the search tree ($\mathcal{BS}[p] = 0$) we do nothing, otherwise ($\mathcal{BS}[p] = 1$) we create a copy c of constraint $h_{\downarrow p}$ and of the decision variables it constrains. Then we enforce GAC on c using $\mathcal{A}$ (line 5). Recall that $h_{\downarrow p}$ is a classical constraint, so we can enforce consistency by using standard propagation techniques. At line 6, if GAC produces a domain wipeout in c at line 5, we set the bit $\mathcal{BS}[p]$ to 0 and, at line 8, we check if the remaining scenarios can provide an overall probability that exceeds θ_h. If they cannot, we backtrack. On the other hand, if GAC does not produces a domain wipeout in c at line 5, at line 10 for each i such that $\mathcal{PT}[i]$ is constrained by $h_{\downarrow p}$, we iterate through the domain of the copy of the decision variable. For each value v that has been pruned when we enforced GAC on c at line 5, we set the respective bit $\mathcal{VBS}[v, \mathcal{PT}[i], p]$ to zero (line 10). The remaining lines of the algorithm are identical to those described in Algorithms 1 and 2.

To summarize, in Algorithm 3 backtrack may occur at line 9, informally speaking if in "too many" scenarios we have observed a domain wipeout, or at line 27, if we observe a domain wipeout after pruning value v from $D(\mathcal{PT}[i])$.

Theorem 8. *Algorithm 3 is a sound filtering algorithm.*

Proof *(sketch).* **Soundness.** A value is either pruned at line 27 or backtracking occurs at line 8. The pruning that happens at line 27 is similar to the one that happens in the non-incremental algorithm. $g[k]$ becomes strictly smaller than θ_h when we replace $\max[i]$ in $g[k]$ with $f[i, v]$. Indeed $g[k]$ is thus equal to $f[i, v] + \sum_{j \in M_k, j \neq i} \max(j)$ which makes Proposition 1

true. The pruned value v is inconsistent. Due to the identity introduced in Eq. (1), when backtracking occurs (at line 8) the probability associated with scenarios for which an inconsistency has been detected amounts to a value greater than $1-\theta_h$. Therefore there exists no policy tree $\mathcal{T}$ for which

$$\sum_{p\in\Psi:\, \tilde{h}_{\downarrow p}\in h_{\downarrow p}} \Pr\{arcs(p)\} \geqslant \theta_h. \qquad \square$$

As in the previous case, we can call Algorithm 3 until we reach a fixed-point and no further pruning is done. We also denote as Algorithm 3 this modified algorithm.

Theorem 9. *Algorithm 3 runs in $O(|\Omega|\cdot a\cdot \mathcal{N}^3\cdot d^3)$ time and in $O(p+\mathcal{N}\cdot d\cdot|\Omega|)$ space where a is the time complexity of $\mathcal{A}$, p is its space complexity, and d is the maximum domain size.*

Proof *(sketch).* **Time complexity.** The proof is identical to that provided for Algorithm 1 except for the fact that now $f[i,v]$ is functionally dependent on $\mathcal{BS}$ and $\mathcal{VBS}$. Therefore, every time $f[i,v]$ is used, its computation requires at worst $|\Omega|$ iterations. Since we compute $f[i,v]$ for each value v in the domain of each decision variable $\mathcal{PT}[i]$ (line 15), the overall time complexity is now in $O(|\Omega|\cdot a\cdot\mathcal{N}^3\cdot d^3)$ time.

Space complexity. As in Algorithm 2, the space complexity is dominated by the size of $\mathcal{BS}$, of $\mathcal{PT}$ and by the space complexity of $\mathcal{A}$. $\mathcal{BS}$ has size $|\Omega|$, in fact we store one bit per scenario. Nevertheless, now we also store a bit set of size $|\Omega|$ for each value v in the domain of each decision variable $\mathcal{PT}[i]$. $\mathcal{PT}$ requires $\mathcal{N}\cdot d$ space whereas $\mathcal{A}$ requires p space. Therefore, the algorithm runs in $O(p+(\mathcal{N}\cdot d+1)\cdot|\Omega|)$ space. $\square$

8. Benchmark problems

In this section we introduce a number of benchmark problems used in our experiments. For each problem we provide the problem definition, a set of instances that will be used in our computational experiments and some considerations on the advantages brought by our novel approach in terms of modeling expressiveness.

8.1. Random stochastic CSPs (RSCSP)

We introduce a number of randomly generated SCSPs.

8.1.1. Problem definition

The SCSPs considered feature five chance constraints over 4 integer decision variables, $x_1,\ldots,x_4$ and 8 stochastic variables, $s_1,\ldots,s_8$.

There are five chance constraints in the model, the first embeds an equality,

$$c_1\colon \Pr\{x_1s_1+x_2s_2+x_3s_3+x_4s_4=80\}\geqslant\alpha,$$

the second and the third embed inequalities,

$$c_2\colon \Pr\{x_1s_5+x_2s_6+x_3s_7+x_4s_8\leqslant 100\}\geqslant\beta,$$

$$c_3\colon \Pr\{x_1s_5+x_2s_6+x_3s_7+x_4s_8\geqslant 60\}\geqslant\beta.$$

The fourth chance constraint embeds again an inequality, but in this case the constraint is defined over a subset of all the decision and random variables in the model:

$$c_4\colon \Pr\{x_1s_2+x_3s_6\geqslant 30\}\geqslant 0.7.$$

Finally, the fifth chance constraint embeds an equality also defined over a subset of all the decision and random variables in the model:

$$c_5\colon \Pr\{x_2s_4+x_4s_8=20\}\geqslant 0.05.$$

We considered 3 possible stage structures. In the first stage structure we have only one stage, $\langle V_1,S_1\rangle$, where $V_1=\{x_1,\ldots,x_4\}$ and $S_1=\{s_1,\ldots,s_8\}$. In the second stage structure we have two stages, $\langle V_1,S_1\rangle$ and $\langle V_2,S_2\rangle$, where $V_1=\{x_1,x_2\}$, $S_1=\{s_1,s_2,s_5,s_6\}$, $V_2=\{x_3,x_4\}$, and $S_2=\{s_3,s_4,s_7,s_8\}$. In the third stage structure we have four stages, $\langle V_1,S_1\rangle$, $\langle V_2,S_2\rangle$, $\langle V_3,S_3\rangle$, and $\langle V_4,S_4\rangle$, where $V_1=\{x_1\}$, $S_1=\{s_1,s_5\}$, $V_2=\{x_2\}$, $S_2=\{s_2,s_6\}$, $V_3=\{x_3\}$, $S_3=\{s_3,s_7\}$, and $V_4=\{x_4\}$, $S_4=\{s_4,s_8\}$.

Constraints:
(1) $\Pr\{\text{alldiff}(y_1, y_2, \ldots, y_8)\} \geqslant \theta$
(2) $y_i = x_i + d_i \quad \forall i \in 1, \ldots, 8$
Decision variables:
$x_i \in \{1, 3, 5, 7, 11, 13, 15\} \quad \forall i \in 1, \ldots, 4$
$x_i \in \{2, 4, 6, 8, 10, 12, 14\} \quad \forall i \in 4, \ldots, 8$
$y_i \in \{1, \ldots, 15\} \quad \forall i \in 1, \ldots, 8$
Random variables:
$d_i \rightarrow$ delay of plane $i \quad \forall i \in 1, \ldots, k$
Stage structure:
$V_1 = \{x_1, x_2, x_3, x_4\}$
$V_2 = \{y_1, y_2, y_3, y_4, x_5, x_6, x_7, x_8\}$
$V_3 = \{y_5, y_6, y_7, y_8\}$
$S_1 = \{d_1, d_2, d_3, d_4\}$
$S_2 = \{d_5, d_6, d_7, d_8\}$
$S_3 = \{\}$
$L = [\langle V_1, S_1\rangle, \langle V_2, S_2\rangle, \langle V_3, S_3\rangle]$

Fig. 5. A three-stage SCSP for the chance-constrained plane landing scheduling.

8.1.2. Instance generation

The decision variable domains are: $D(x_1) = \{5, \ldots, 10\}$, $D(x_2) = \{4, \ldots, 10\}$, $D(x_3) = \{3, \ldots, 10\}$, and $D(x_4) = \{6, \ldots, 10\}$. The domains of stochastic variables s_1, s_3, s_5, s_7 comprise 2 integer values each. The domains of stochastic variables s_2, s_4, s_6, s_8 comprise 3 integer values each. The values in these domains have been randomly generated as uniformly distributed in $\{1, \ldots, 5\}$. Each value appearing in the domains of random variables s_1, s_3, s_5, s_7 is assigned a realization probability of $\frac{1}{2}$. Each value appearing in the domains of random variables s_2, s_4, s_6, s_8 is assigned a realization probability of $\frac{1}{3}$. Parameters α and β take values in $\{0.005, 0.01, 0.03, 0.05, 0.07, 0.1\}$ and $\{0.6, 0.7, 0.8\}$, respectively. In total, we therefore consider 18 different possible configurations for the parameters α and β. For each of these configurations, we generate 15 different probability distributions — i.e. sets of values in the domains — for the random variables in our model. These probability distributions were divided in three groups and employed to generate 5 single-stage problems, 5 two-stage problems and 5 four-stage problems. Therefore the test bed comprised, in total, 270 instances.

8.1.3. Modeling expressiveness

It should be noted that the approach discussed in [26], i.e. SBA, requires several auxiliary constraints and decision variables to model the problems above. In contrast, by using our novel modeling approach, we obtain significantly more compact formulations. More specifically, the single-stage problem is modeled, in [26], using 6484 decision variables and 6485 constraints, while GCC — our approach — requires only 4 decision variables and 5 chance constraints; this is mainly due to the fact that, in addition to the 4 decision variables required to build the policy tree, SBA introduces 1296 binary decision variables for each of the 5 chance constraints in the model; furthermore, SBA also introduces 1297 reification constraints for each chance constraint in the model, similarly to what is shown in Example 1 (Fig. 2). The two-stage problem is modeled by SBA using 6554 decision variables (74 for the policy tree and 6480 binary decision variables) and 6485 constraints, while GCC requires only 74 decision variables and 5 chance constraints; finally, the four-stage problem is modeled by SBA using 6739 decision variables and 6485 constraints, while GCC requires only 259 decision variables and 5 chance constraints.

8.2. Plane Landing Scheduling Problem (PLSP)

Our second benchmark problem is the SCSP in Fig. 5. This model is fairly simple, still it captures an important practical problem: the control of landing conflicts for P planes on a single runway under stochastic arrival delays.

8.2.1. Problem definition

Consider a set of L_i available landing slots for plane i. Decision variable x_i represents the landing slot — for instance a 15 minutes time interval — assigned to plane i, the random variable d_i represents the random delay of plane i. Decision variables $x_1, x_2, \ldots, x_8$ are partitioned in two stages, $V_1 = \{x_1, x_2, \ldots, x_4\}$ and $V_2 = \{x_5, x_2, \ldots, x_8\}$. Similarly, the respective random delays d_i are also partitioned in two decision stages. This means that once the delays of the first 4 planes have been observed it is possible to act consequently and choose the most appropriate recourse action that is available in the policy tree.

Enforcing constraint (1), under the stage structure described in Fig. 5 means ensuring that the probability of observing a landing conflict — i.e. two planes that land within the same time slot — remains below the specified threshold $1 - \theta = 0.05$ (i.e. $\theta = 0.95$). More specifically x_i denotes the "planned" landing slot, while y_i represents the "realized" arrival time. x_i is decided at stage 1 (for $i = 1, \ldots, 4$) or at stage 2 (for $i = 5, \ldots, 8$), before the actual delay of plane i, $\bar{d}_i$, is observed. Conversely, y_i represents the realized landing time, which is equal to the planned arrival time x_i plus the realized delay $\bar{d}_i$. Decision variables y_i are fixed only after the realized delays are known. The domain of y_i ranges over the total number T of available landing slots, for instance this may be a whole 8-hour working day planning horizon.

In every possible scenario $\omega \in \Omega$ of the policy tree associated with the SCSP in Fig. 5 a delay $\bar{d}_i^{\omega}$ is associated with each plane i. If for two planes i and j the "realized" delays in scenario ω are such that $x_i + \bar{d}_i^{\omega} = x_j + \bar{d}_j^{\omega}$, that is $y_i^{\omega} = y_j^{\omega}$, then we have a landing conflict in scenario ω. A feasible landing plan is therefore a satisfying policy tree that guarantees a probability of conflict lower than $1 - \alpha$.

The reader should be aware of the limitations of this simple example. For instance, in a satisfying policy tree for our model some planes in V_1 may be scheduled at a time that comes after the scheduled time for planes in V_2. A realistic model should prevent these situations by forcing planes in V_1 to be scheduled at earlier time slots. Furthermore, the model presented can be made more realistic by adding more runways, slots and airports, by modeling the connection times and by therefore providing a full schedule that guarantees a given service level. Discussing the complete problem is out of the scope of this work, since our objective here is to demonstrate a practical application area for the stochastic alldiff constraint and to investigate the filtering effectiveness of our strategies in a proof-of-concept model. However, the simple model we presented already gives a clear idea of how relevant a constraint such as the stochastic alldiff is for practical applications. We leave the investigation of a complete model for plane landing scheduling as a possible direction for future research.

8.2.2. Instance generation

We consider $\theta \in \{0.95, 0.90, 0.85, 0.80, 0.75, 0.70\}$ and 5 sets of different probability distributions for each random variable d_i, these distributions have been randomly generated by selecting two possible integer delays uniformly distributed in $\{1, \ldots, 4\}$ to each of which a realization probability equal to 0.5 is then assigned. The available landing slots (decision variable domains) go from 1 to 15, the maximum landing time $T = 19$, in fact the maximum observable delay is 4. Note that, as shown in Fig. 5, we consider domains with holes for each decision variables in order to let the GAC algorithm exploit the structure of the problem. In total, we therefore generated 30 different instances.

8.2.3. Modeling expressiveness

The policy tree for the model in Fig. 5 comprises 2116 decision variables: 4 at the root node (x_i, $i \in \{1, \ldots, 4\}$), 64 at the first stage (x_i, $i \in \{5, \ldots, 8\}$) and 2048 at the third stage (y_i, $i \in \{1, \ldots, 8\}$).

The reader should be aware that bidirectional implications such as

$$s = 1 \leftrightarrow \text{alldiff}(x, y, z)$$

involving global constraints are not allowed in most constraint solvers; for instance Choco [15] does not support this construct. One of the solvers that effectively supports this construct is Minion [10]. In this solver, an SBA model would require 256 auxiliary binary variables for encoding chance constraint (1).

For those solvers that do not support the above construct, in order to model the SCSP in Fig. 5 by using the approach in [26], it is often possible to adopt a decomposition for the alldiff constraint. However, the associated model is clearly not only unreadable, but also extremely inefficient in terms of propagation effectiveness, in fact the structure of the problem is totally lost and a significant number of auxiliary binary variables have to be employed in the scenario-based model to decompose the alldiff constraint in each scenario. These variables add up to the 256 required to encode the chance constraint.

We observe again that, by using our novel modeling approach, we obtain significantly more compact model formulations than the state-of-the-art approach in [26].

8.3. Stochastic Knapsack Problem (SKP)

Our last benchmark problem is the stochastic knapsack problem [12] — a known problem in stochastic constraint optimization.

8.3.1. Problem definition

A subset of k items must be chosen, given a knapsack of size c into which to fit the items. Each item i, if included in the knapsack, brings a stochastic profit r_i. Also the size ω_i of each item is stochastic and it is not known at the time the decision has to be made. Nevertheless, we assume that the decision maker knows the probability mass functions PMF(ω_i) and PMF(r_i) [13], for each $i = 1, \ldots, k$. The probability of the plan not exceeding the capacity C of the knapsack should be greater than or equal to a given threshold θ. The objective is to find the knapsack that maximizes the expected profit.

We consider both the single and the multi-stage formulation of the problem. In the single-stage formulation, objects are selected before any of the respective profits or weights have been observed. In the multi-stage formulation, items are considered sequentially, starting from item 1 up to item k. In other words, first we take the decision of inserting or not a given object into the knapsack, then we immediately observe its weight, which is a random variable, before any further item is taken into account.

In Fig. 6 we provide a stochastic constraint programming formulation for the SKP exploiting global chance constraints. In this model, the objective function maximizes z, that is the expected total profit brought by the objects selected in the knapsack — those for which the binary decision variable x_i is set to 1. This expectation is computed in chance constraint (1). Chance constraint (2) ensures that the capacity C is not exceeded with a probability of at least θ. The model in Fig. 7 is a single-stage model in which we first select all the objects we want to include in the knapsack and then we observe their weights and profits.

Objective:
$\max z$
Subject to:
(1) $z = \mathbb{E}[\sum_{i=1}^{k} r_i x_i]$
(2) $\Pr\{\sum_{i=1}^{k} \omega_i x_i \leqslant C\} \geqslant \theta$
Decision variables:
$x_i \in \{0, 1\} \quad \forall i \in 1, \ldots, k$
$z \in \mathbb{R}$
Random variables:
$\omega_i \rightarrow$ item i weight $\quad \forall i \in 1, \ldots, k$
$r_i \rightarrow$ item i profit $\quad \forall i \in 1, \ldots, k$
Stage structure:
$V_1 = \{x_1, x_2, \ldots, x_k, z\}$
$S_1 = \{r_1, r_2, \ldots, r_k, \omega_1, \omega_2, \ldots, \omega_k\}$
$L = [\langle V_1, S_1 \rangle]$

Fig. 6. Stochastic constraint programming formulation for the single-stage SKP.

Stage structure:
$V_1 = \{x_1, z\} \ \forall i \in 1, \ldots, k$
$V_i = \{x_i\} \ \forall i \in 2, \ldots, k$
$S_i = \{r_i, \omega_i\} \ \forall i \in 1, \ldots, k$
$L = [\langle V_1, S_1 \rangle, \langle V_2, S_2 \rangle, \ldots, \langle V_k, S_k \rangle]$

Fig. 7. Stage structure for the multi-stage SKP.

In Fig. 7 we provide an alternative stage structure, that can be used in place of the stage structure in Fig. 6 to formulate the multi-stage SKP. The model now comprises, in the stage structure L, multiple decision stages that alternate decisions and observations according to the arrival sequence of the objects. In practice, in an optimal policy for the multi-stage SKP an object may be selected or not, depending on the realized weights for previous objects.

Please note that Choco [15], the underlying solver we adopt for our algorithms, provides native support for real valued decision variables (object `RealVar`), therefore it is straightforward to define and handle variable z during search and propagation. Furthermore, $\mathbb{E}[\sum_{i=1}^{k} r_i x_i]$ is an expression involving expected values. These expressions can in principle be handled using a generic reformulation as follows. Let $\mathbb{E}[\langle \text{exp} \rangle]$ denote the expected value of $\langle \text{exp} \rangle$. Recall that Ψ denote the set of all distinct paths of a policy tree in the SCSP of interest. Since we assume that the support of random variables is finite, it follows that the expected value can be easily reduced to a fully deterministic expression

$$\mathbb{E}\big[\langle \text{exp} \rangle\big] = \sum_{p:\ p \in \Psi} \langle \text{exp} \rangle_{\downarrow p} \cdot \Pr\big\{arcs(p)\big\},$$

where $\langle \text{exp} \rangle_{\downarrow p}$ is the deterministic expression obtained by replacing every random variable in $\langle \text{exp} \rangle$ with the respective deterministic value this variable takes in scenario $arcs(p)$ and every decision variable in $\langle \text{exp} \rangle$ with the respective copy $\mathcal{PT}[i]$ associated with path p. Note, however, that Choco [15] does not allow expressions of mixed types, i.e., float and integer types. Therefore, we implement a simple filtering algorithm that handles expected values expression which should compute a real value, but the expression involves some integer values as well. This filtering algorithm is discussed in Appendix A.

8.3.2. Instance generation

We consider a number of randomly generated instances for the single and multi-stage SKP. The SCSPs considered feature a single chance constraints over 4 integer decision variables, $x_1, \ldots, x_4$, and 4 stochastic variables, $\omega_1, \ldots, \omega_4$, representing object weights. The decision variable domains are: $D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{0, 1\}$. The domains of stochastic variables $\omega_1, \ldots, \omega_4$ comprise 2 integer values each. The values in these domains have been randomly generated as uniformly distributed in $\{1, \ldots, 100\}$. Furthermore, the model also comprises 4 stochastic variables, $r_1, \ldots, r_4$ representing the random profit brought by a given object, once it has been selected in a knapsack. Also the domains of stochastic variables $r_1, \ldots, r_4$ comprise 2 integer values each randomly generated as uniformly distributed in $\{1, \ldots, 100\}$. Each value appearing in the domains of random variables is assigned a realization probability of $\frac{1}{2}$. We generated 5 different random instances, then for each of these instances we consider θ ranging in $\{0.95, 0.90, 0.85, 0.80, 0.75\}$ and C ranging in $\{300, 250, 200, 150, 100\}$. This produced a test bed of 125 instances. We consider 2 possible stage structures: in the first we have only one stage, $\langle V_1, S_1 \rangle$, where $V_1 = \{z, x_1, \ldots, x_4\}$ and $S_1 = \{r_1, \ldots, r_4, \omega_1, \ldots, \omega_4\}$; in the second we have two stages, $\langle V_1, S_1 \rangle$ and $\langle V_2, S_2 \rangle$, where $V_1 = \{z, x_1, x_2\}$, $S_1 = \{r_1, r_2, \omega_1, \omega_2\}$, $V_2 = \{x_3, x_4\}$, and $S_2 = \{r_3, r_4, \omega_3, \omega_4\}$. The complete test bed therefore comprises 250 instances.

8.3.3. Modeling expressiveness

It is clear that, under the first stage structure, the policy tree comprises only 4 binary decision variables and the real valued variable z; under the second stage structure, it comprises 34 binary decision variables and the real valued variable z. Of course, as discussed in the previous sections, the SBA model requires a much larger number of variables to encode the chance constraints in the model. Roughly, the additional number of binary variables required by SBA is proportional to the number of scenarios and of chance constraints in the model, regardless of the stage structure. In this case, since we have 8 binary discrete random variables, the number of scenarios amounts to 2^8. Therefore the SBA model includes at least 256 auxiliary binary variables for the chance constraint enforcing the capacity restriction, and 256 auxiliary integer variables for computing the expected cost. A comparable number of auxiliary constraints is also introduced. We stress once more that by using our novel modeling approach we obtain significantly more compact model formulations than the state-of-the-art approach in [26].

9. Computational experience

In this section we discuss our computational experience aimed at answering the following questions:

(1) Does the new approach based on the proposed filtering algorithms bring any benefit in terms of pruning compared to the state-of-the-art approach?
(2) Does the new approach based on the proposed filtering algorithms bring any benefit in terms of search efficiency compared to the state-of-the-art approach?
(3) What effect can we observe when we vary the level of consistency of algorithm $\mathcal{A}$?
(4) Is the new approach based on the proposed filtering algorithms more scalable?

All the experiments were performed on an Intel Core 2 Duo 1.86 GHz with 2 GB RAM. The solver used for our test is Choco 1.2 [15], a Java open source CP solver. Variable and value selection heuristics were selected empirically among the following ones made available in Choco [15] ("min domain", "dom over den degree", "dom over degree", "most constrained"). The combination adopted for RSCSP and SKP is the one that gave better results for SBA. For SPLSP, since we do not compare against SBA, we arbitrarily selected a "min domain" heuristic for variable selection and then we analyzed the impact of different value selection heuristics on search performances.

9.1. Pruning effectiveness

Consider the RSCSPs introduced in Section 8. We compare the effectiveness of the filtering performed by SBA and GCC (Algorithm 1).

The propagation strategy discussed in Section 6 requires an existing propagator $\mathcal{A}$ for the deterministic constraints. Since the only constraints appearing in the RSCSPs above are linear (in)equalities, we employ a simple bounds consistency procedure for linear (in)equalities implemented in Choco 1.2 [15].

In this experiment, we only consider 90 two-stage feasible instances of the 270 instances of RSCSPs randomly generated according to the strategy discussed in Section 8 (5 different probability distributions for the random variables and 18 different configurations for parameters α and β). We generate a solution for each of these instances. Then we randomly pick subsets of the decision variables in the problem, we assign them to the value they take in this solution, we propagate according to SBA and GCC, respectively, and we compare the percentage of values pruned by each of these two approaches.

In Fig. 8 we show the results of this comparison, which is performed for a number of decision variables assigned that ranges from 0% — this corresponds to a root node propagation — to 90% of the decision variables that appear in the policy tree.

In the graph, for each percentage of decision variables assigned, we report — in percentage on the total amount of values in the initial decision variable domains — the minimum, the maximum, and the average number of values pruned from the domains. As it appears from the graph, if we consider the minimum percentage of values pruned by the two approaches, GCC always achieves a stronger pruning than SBA in the worst case. Furthermore, as the maximum percentage of values pruned reported in the graph witnesses, GCC is able to achieve a much stronger pruning than SBA in the best case. On average, GCC always outperforms SBA, by filtering up to 8.64% more values when 60% of the decision variables are assigned and at least 3.11% more values at the root node.

The reader should note that the filtering effectiveness for a given algorithm $\mathcal{A}$ does not vary for Algorithms 1, 2 and 3. Therefore only the computational efficiency (i.e. number of calls to algorithm $\mathcal{A}$) changes. We will investigate this further in the next section.

9.2. Search efficiency

In the experiments presented so far, by using our novel approach, we outperform the state-of-the-art approach in [26] in terms of pruning. We now investigate if this is reflected in gains in terms of search efficiency. We consider two benchmark problem: a feasibility problem (RSCSPs) and an optimization problem (SKP). Both these problems have been introduced in

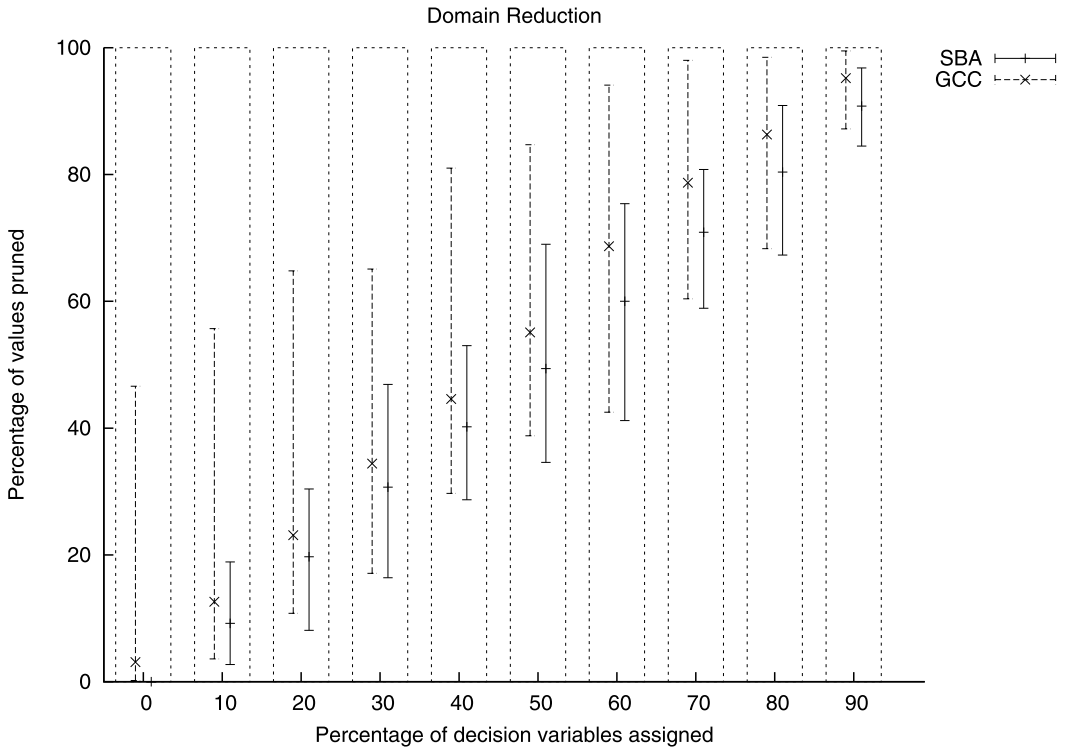

Fig. 8. Effectiveness of the filtering performed by SBA and GCC (Algorithm 1).

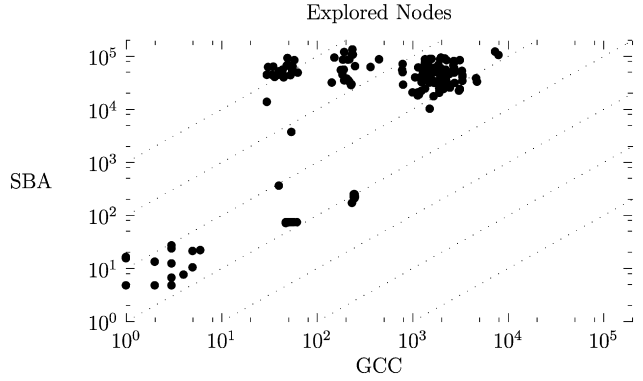

Fig. 9. The graph compares SBA and GCC (Algorithm 1) in terms of explored nodes for the 270 instances in our test bed. Axes are in logarithmic scale.

Section 8. We now show that, by using our novel modeling approach (GCC) in concert with the non-incremental propagation strategy in Algorithm 1, we outperform the state-of-the-art modeling approach in [26] (SBA) in terms of runtimes and explored nodes. Furthermore, we show that incremental filtering (Algorithms 2 and 3) is computationally more efficiency than non-incremental filtering (Algorithm 1). These gains in efficiency also increase as we increase the number of stages.

9.2.1. RSCSPs

In order to assess search efficiency, we compared our approach (GCC) — which models the discussed SCSPs using five global chance constraints, one for each chance constraint in the model — against the deterministic equivalent CSPs generated using the state-of-the-art scenario-based approach in [26] (SBA).

In our comparative study we consider the 270 instances of RSCSPs discussed in Section 8. The variable selection heuristic used during the search is the *domain over dynamic degree* strategy, while the value selection heuristic selects values from decision variable domains in *increasing* order. To each instance we assign a time limit of 240 seconds for running the search. The computational performances of Algorithm 1 and SBA are compared in Figs. 9 and 10. Runtimes for Algorithms 1 and 2

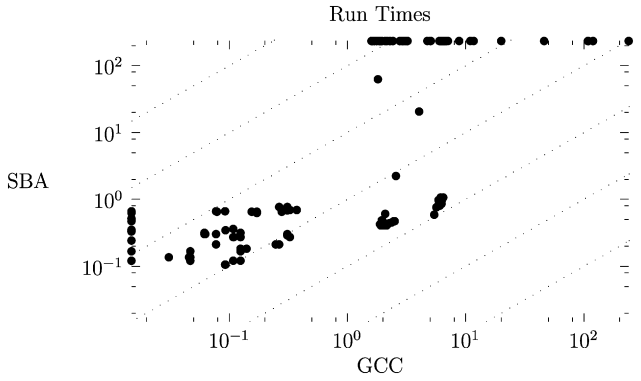

Fig. 10. The graph compares the run time performance of SBA and GCC (Algorithm 1) for the 270 instances in our test bed. Axes are in logarithmic scale.

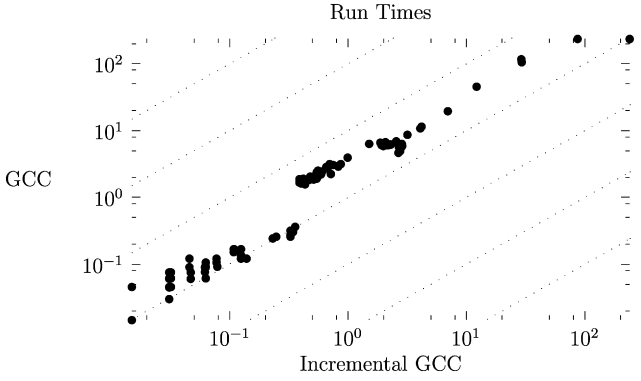

Fig. 11. The graph compares the run time performance of non-incremental GCC (Algorithm 1) and incremental GCC (Algorithm 2) for the 270 instances in our test bed. Axes are in logarithmic scale.

are compared in Fig. 11. A more detailed overview on our computational experience is given in Fig. 12, which presents a comprehensive set of boxplots[3] for our experiments.

The results show that GCC (Algorithm 1) solved all the instances that SBA could solve within the time limit. In contrast, SBA was often not able to solve — within the given time limit — instances that GCC could solve in a few seconds. More specifically, both GCC and SBA could solve 90 of 90 1-stage instances; on average GCC explored roughly 5 times less nodes and was about 3.34 times faster than SBA for these instances. GCC could solve 45 of 90 2-stage instances, while SBA could only solve 18 of them; on average GCC explored roughly 36 times less nodes and was about 15 times faster than SBA for these instances. Finally, GCC could solve 31 of 90 4-stage instances, while SBA could only solve 10 of them; on average GCC explored roughly 35 times less nodes and was about 20 times faster than SBA for these instances. Incremental GCC (Algorithm 2) could solve: 90 of 90 1-stage instances, on average it was about 3.90 times faster than SBA and 1.16 times faster than GCC for these instances; 45 of 90 2-stage instances, on average it was about 58 times faster than SBA and 3.87 times faster than GCC for these instances; and 32 — therefore one instance more than GCC — of 90 4-stage instances, on average it was about 29 times faster than SBA and 2.70 times faster than GCC for these instances.

9.2.2. SKP

We consider the 250 instances of SKP generated as discussed in Section 8. Since the only constraint appearing in the SKP is, once more, a linear inequality, we employ also in this case the simple bounds consistency procedure for linear (in)equalities implemented in Choco 1.2 [15] as existing propagator $\mathcal{A}$. The variable selection heuristic used during the search is the *min domain* strategy, while the value selection heuristic selects values from decision variable domains in *decreasing* order.

We compare the computational performances of the incremental versions of the filtering algorithms (Algorithms 2 and 3) and SBA. Note that non-incremental version of the filtering algorithms (Algorithm 1) is not included in the experiments as it is always outperformed by the incremental versions. Limits were imposed, during search, for run time and explored nodes. More specifically, since SKP is an optimization problem and the solution time per instance tends to be higher than the one observed for RSCSP, we limited the run time to 4000 seconds and the search space to 10,000,000 nodes.

[3] A box plot [27,17] is a convenient way of graphically depicting groups of numerical data through their five-number summaries: the smallest observation, lower quartile, median, upper quartile, and largest observation. A boxplot also indicates which observations might be considered outliers.

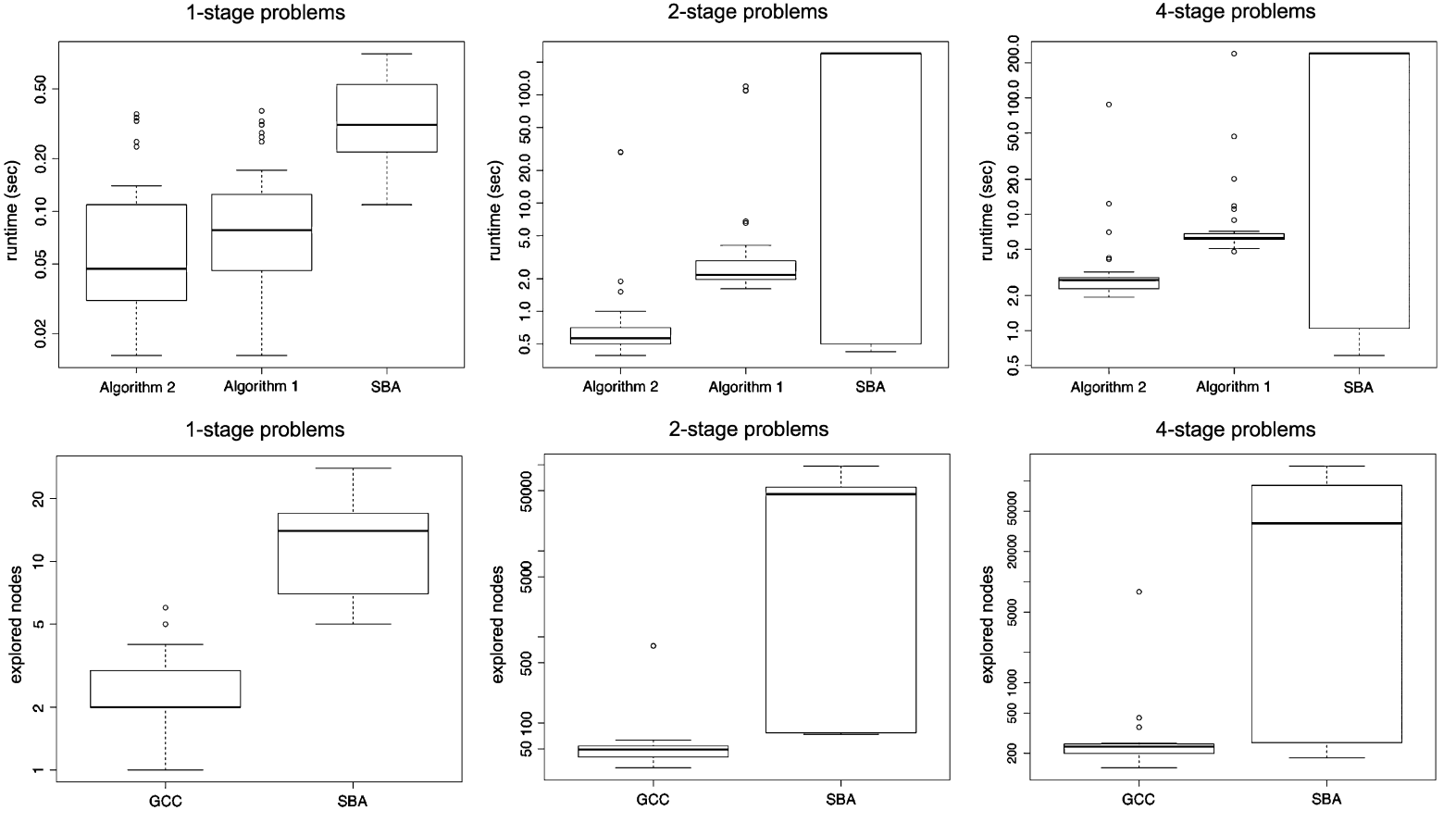

Fig. 12. Randomly generated SCSPs. Boxplots for run time (in seconds) and explored nodes in the test bed considered. For the 2-stage and 4-stage instances the boxplots only refer to the subset of instances that could be solved by GCC. The y-axis is displayed in logarithmic scale.

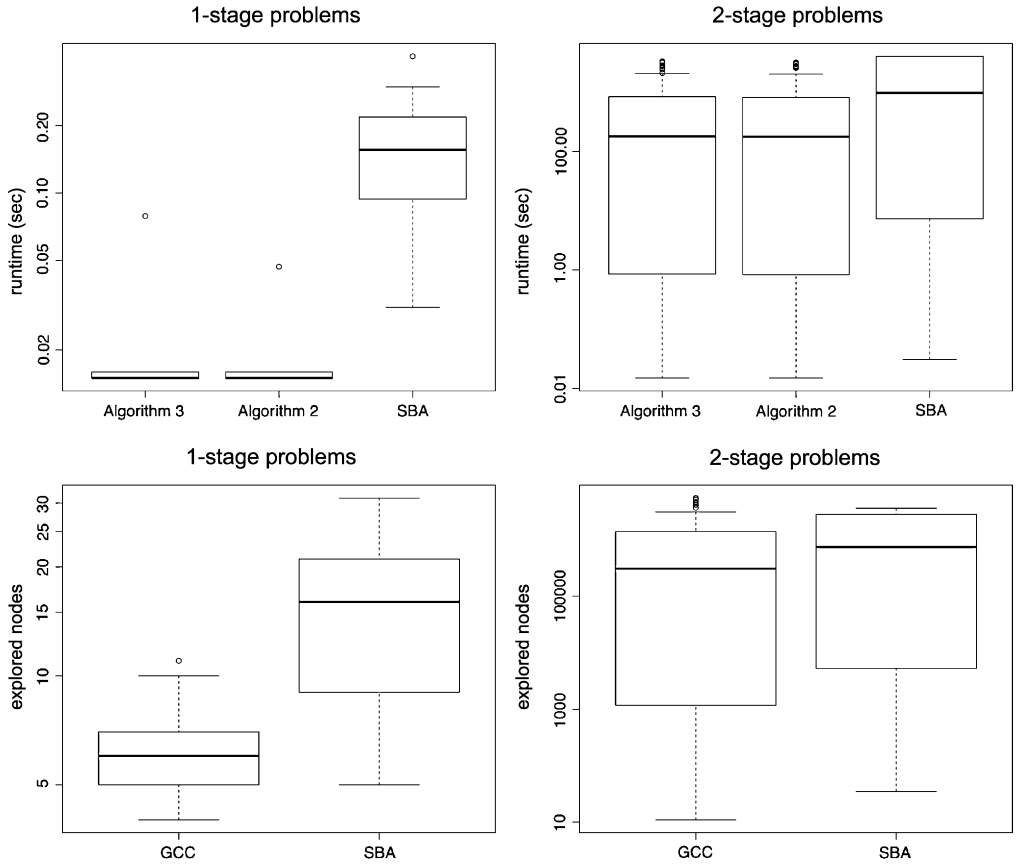

Fig. 13. Stochastic knapsack problem. Boxplots for run time (in seconds) and explored nodes in the test bed considered. For the 2-stage instances the percentiles only refer to the 116 instances that could be solved by GCC. The y-axis is displayed in logarithmic scale.

The runtimes obtained by Algorithm 2 are comparable to those obtained by Algorithm 3. This ought to be expected, in fact the model comprises only a chance constraint embedding a linear inequality; this implies that bounds consistency is sufficient for guaranteeing GAC in each scenario; therefore tracking disentailment at value level cannot bring any benefit. Nevertheless, it is interesting to note that the memory requirements of Algorithm 3 does not significantly impact computational performances.

In the test bed considered, the incremental algorithms and SBA could solve, within the given time and node limits, 125 of 125 1-stage instances; on average both the incremental algorithms explored 2.4 times less nodes and were about 10 times faster than SBA for these instances. The incremental algorithms could solve 116 of 125 2-stage instances; of these 116 instances, SBA only solved 84. On average, both incremental algorithms explored at least 3.86 times less nodes — the "at least" refers to the fact that SBA hit the time and node limits imposed for some instances — and were at least about 7 times faster than SBA for these instances. None of the remaining 9 instances that the incremental algorithms could not solve was solved by SBA within the allocated time and node limits. Over the set of 116 2-stage instances that the incremental algorithms could solve to optimality, SBA hit the imposed run time limit for about 33% of the instances. In Fig. 13 we report boxplots for run times and explored nodes over the test bed considered. The above experiments show that our incremental filtering strategies are computationally more efficient than the state-of-the-art approach in [26].

9.3. Comparing the incremental algorithms

The previous experiments show that both incremental algorithms have similar performances for the SKP. We now investigate further the relative performance of the incremental algorithms.

We analyzed the incremental filtering algorithms when different degrees of consistency — namely, BC and GAC — are enforced by the parameterizing algorithm $\mathcal{A}$ in Algorithms 2 and 3. We use the PLSP as our benchmark problem.

The experiments show that these two algorithms are in general incomparable, and that their effectiveness depends upon the consistency level achieved by algorithm $\mathcal{A}$ that is used, on the heuristics and, clearly, on the problem being investigated.

We consider the 30 instances of PLSP generated as discussed in Section 8. We computed a solution for each of these instances and we analyzed the impact of the pruning when a given percentage of the decision variables in the model have been assigned to their value in the solution computed. The results are shown in Fig. 14.

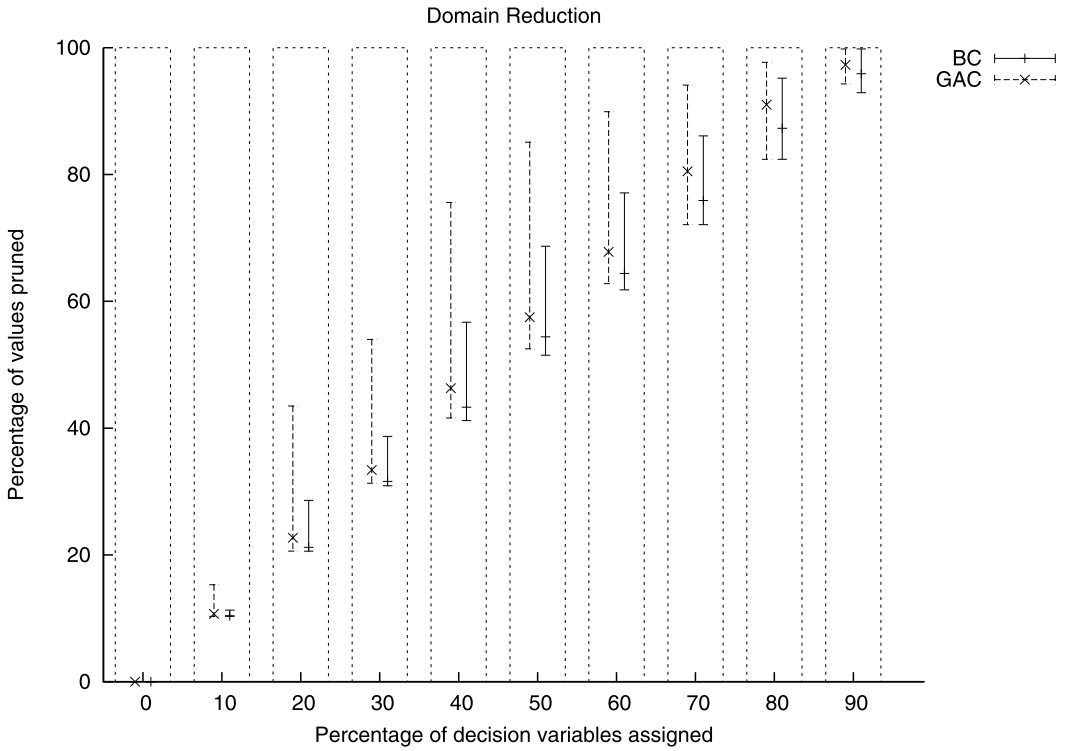

Fig. 14. Effectiveness of the filtering when different degrees of consistency are enforced by the parameterizing algorithm $\mathcal{A}$: GAC vs BC for stochastic alldiff.

In the graph, for each percentage of decision variables assigned, we report — as a percentage of the total number of values in the initial decision variable domains — the minimum, the maximum, and the average number of values pruned from the domains over the 30 instances considered. As is apparent from the graph and as expected, if we consider the minimum percentage of values pruned by the two approaches, an algorithm enforcing GAC always achieves a stronger pruning than one enforcing BC in the worst case. Furthermore, as the maximum percentage of values pruned reported in the graph witnesses, an algorithm enforcing GAC is able to achieve a much stronger pruning than one enforcing BC in the best case. On average, the former always outperforms the latter, by filtering on average up to 4.66% more values when 70% of the decision variables are assigned.

Although it is clear, from our previous discussion, that an algorithm $\mathcal{A}$ enforcing a stronger level of consistency leads to more pruning in Algorithms 2 and 3, it is not immediately seen if this brings an effective benefit in terms of runtimes. Indeed, we can have four possible combinations: a lightweight filtering algorithm, i.e. Algorithm 2, combined with algorithm $\mathcal{A}$ enforcing a weak consistency, such as BC; or a memory intensive filtering algorithm, i.e. Algorithm 3, combined with a GAC propagator $\mathcal{A}$; or the two possible intermediate options, that is Algorithm 3 in combination with BC and Algorithm 2 in combination with GAC. We will now investigate this issue.

In what follows, once more we consider the PLSP problem in Fig. 5. Nevertheless, we now fix θ to 0.95 and we generate 50 sets of probability distributions random variables d_i; this is done according to the same strategy previously discussed. We solve each of these 50 instances by using two possible filtering algorithms for the alldiff constraint: the algorithm enforcing GAC in [20] and the algorithm enforcing bound consistency (BC) in [16]; each of these algorithms is used in concert with Algorithms 2 and 3. The variable selection heuristic is the *min domain* strategy, the value selection heuristic selects values from decision variable domains in *increasing* order.

Our computational experience is shown in Fig. 15. According to these results, it is not straightforward to decide which consistency level should be used in concert with one of the algorithms we proposed. When we consider an algorithm $\mathcal{A}$ enforcing GAC, Algorithm 3 generally provides better performances than Algorithm 2 over the test bed presented. Conversely, when we consider an algorithm $\mathcal{A}$ enforcing BC, Algorithm 2 generally provides better performances than Algorithm 3. In fact, Algorithm 2 in concert with BC seems to provide the best performances; nevertheless, it fails to solve 3 over 50 instances in the given limit of 1000 explored nodes. Conversely, if a GAC propagator is employed, both Algorithms 2 and 3 fail to solve only 2 instances, but the runtime spent on each instance grows visibly.

In our experiments, we also considered a different value selection heuristic, which selects values from decision variable domains in *decreasing* order. Under this new heuristic strategy, computational times are sensibly impacted, especially for the case in which $\mathcal{A}$ enforces BC in both Algorithms 2 and 3. Furthermore, if a BC propagator is employed, both Algorithms 2

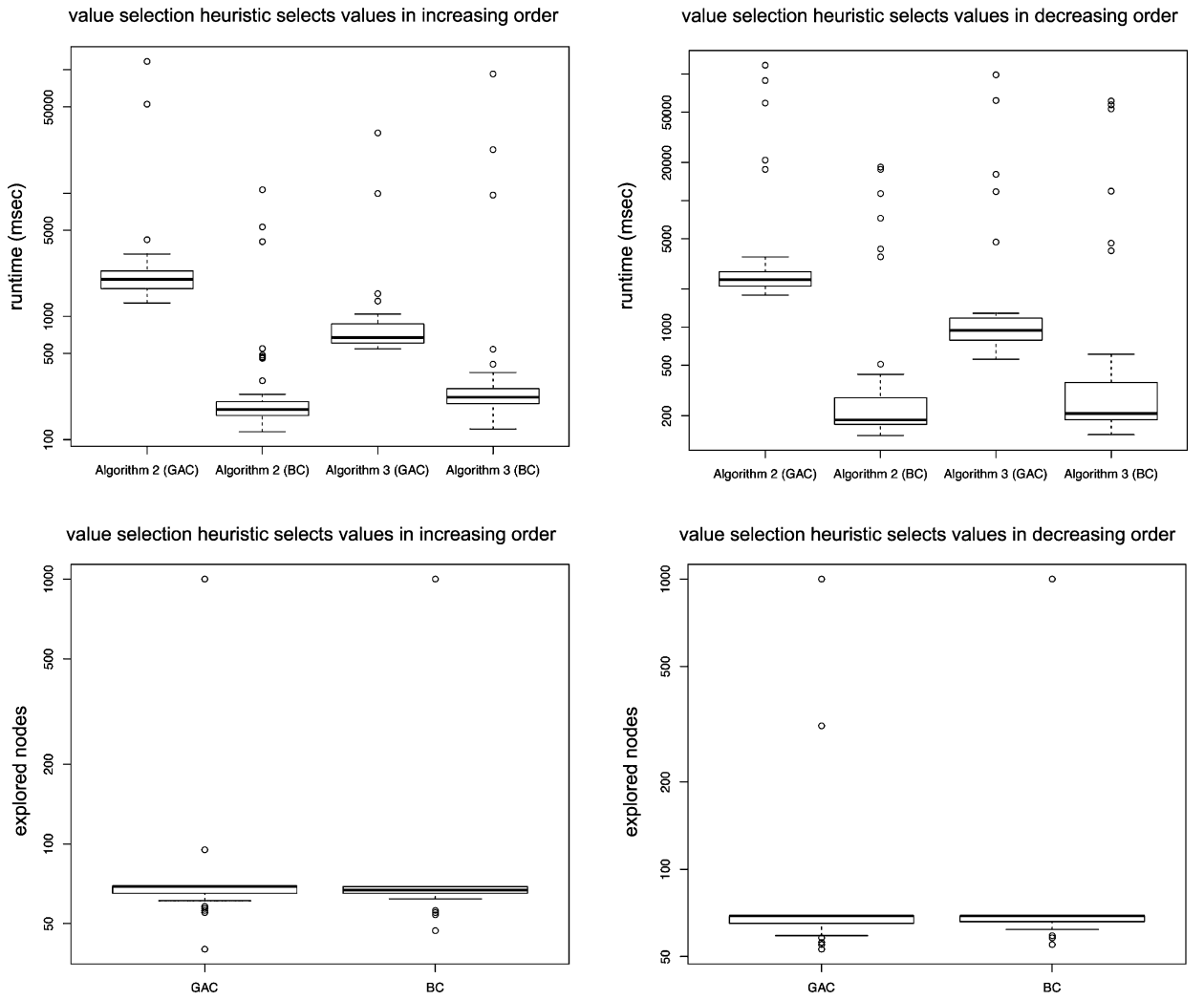

Fig. 15. Plane landing scheduling problem. Comparison between Algorithms 2 and 3 when different degrees of consistency (GAC & BC) are enforced. Boxplots for runtimes and explored nodes over the 50 instances considered in our test bed. Runtimes are in milliseconds, the limit for explored nodes is set to 1000. Two value selection heuristics are considered: increasing and decreasing order. The y-axis is displayed in logarithmic scale.

and 3 fail to solve 6 instances. Conversely, if a GAC propagator is employed, both Algorithms 2 and 3 fail to solve 4 instances. This shows the importance of the use of an adequate heuristic strategy.

Our limited experience revealed that a more lightweight consistency (BC) may payoff, if used in concert with Algorithm 2. Nevertheless, there is the risk of observing high run times for "hard" instances. To overcome this issue, it seems a viable strategy to use a GAC propagator in concert with Algorithm 3.

As a takeaway message, we aim to emphasize that Algorithm 3 tends to use a significant amount of memory, if the number of scenarios is large; in such cases it should be avoided. On the other hand it is the ideal choice for highly combinatorial problems involving a small number of scenarios. Algorithm 2 has low memory requirements and provides competitive performances when used in concert with a BC propagator and a good value and variable selection heuristics.

9.4. Scalability

We finally conducted further experiments on a single-stage SKP comprising 10 objects. 125 instances have been generated for this problem by using the same strategy discussed in Section 8; that is by generating uniformly distributed random values for the probability mass functions of the random profits, weights, and by varying θ and C as discussed. Since each of the 10 objects has 2 possible profits and 2 possible weights, the total number of scenarios is 2^{20}. Choco could not build the scenario based model for any of the 125 instances due to out-of-memory exceptions. The total amount of memory assigned to the VM was the default value for Java 1.6, i.e. a minimum of 2 MB and a maximum of 64 MB. The reader should also be aware that it does not make sense to use Algorithm 3 when the number of scenarios is large; for this reason we did not conduct experiments involving this algorithm. In contrast, Algorithm 2 managed to solve 115 of the 125 instances. The

mean solution time was 51 minutes, the max solution time was 3.24 hours, the median time to solution was 19 minutes. The mean number of explored nodes was 28.5, the max number of explored nodes was 140, the median number of explored nodes was 14. These latter experiments demonstrate the scalability of our approach in the number of scenarios.

10. Related works

A thorough review on hybrid CP/AI/OR approach for decision making under uncertainty is given in [12]. Closely related to our approach are [23,25,24,22]. In these works ad-hoc filtering strategies for handling specific chance constraints are proposed. However, the filtering algorithms presented in both these works are special purpose, incomplete, and do not reuse classical propagators for conventional constraints. Other search and consistency strategies, namely a backtracking algorithm, a forward checking procedure [28] and an arc-consistency [1] algorithm have been proposed for SCSPs. But these present several limitations and cannot be directly employed to solve multi-stage SCSPs as they do not explicitly feature a policy tree representation for the solution of an SCSP. Further extensions to cope with problems involving branching and with multi-objective decision making were discussed in [6]. These extensions only require a minor modification of the original framework. Finally, efforts that try to extend the classical CSP framework to incorporate uncertainty have been influenced by works that originated in different fields, namely *chance-constrained programming* [7] and *stochastic programming* [5]. The Probabilistic CSP [8] represents the first attempt to include random variables, and thus uncertainty, within the CP framework. To the best of our knowledge the first work that tries to create a bridge between Stochastic Programming and Constraint Programming is by Benoist et al. [3]. The idea of employing a scenario-based approach for building up constraint programming models of SCSPs is not novel, since Tarim et al. [26] have already used this technique to develop a fully featured language — Stochastic OPL — for modeling SCSPs. Nevertheless, unlike our approach, the technique in [26], as well as the existing scenario-based reformulation techniques in stochastic programming [5], introduce a significant number of auxiliary binary variables that hinder the search process and that impact the space requirements for both constraint and mathematical programming solvers, respectively. Our work proposes an orthogonal approach to solving SCSPs that do not rely on binary variables and that can easily be integrated with the compilation approach of [26] and with the cost-based filtering techniques in [25] to improve performances. In real-world SCSPs, domains of random variables are typically large and the policy tree tends to explode. We believe that for these problems one has to either: develop a special purpose filtering techniques, if optimality is of concern (see e.g. [23,24]); or adopt some scenario reduction method such as those discussed in [26] (i.e. Latin Hypercube Sampling, Dupacova reduction, etc.) to limit the size of the policy tree in the respective SCSP. In [21], we proposed two novel tools — "Sampled SCSP" and (α, ϑ)-solutions — that allow a decision maker to enforce likelihood guarantees on the quality of the solution obtained when a scenario reduction technique is applied to bound the size of the policy tree. These scenario reduction approaches can be used in problems featuring non-independent random variables and can be applied in synergy with the filtering algorithms discussed in our paper. Alternatively, one may apply the heuristic approach discussed in [18,19], which is based on evolutionary search.

11. Conclusions

We proposed a generic filtering algorithm (Algorithm 1) for global chance constraints. This algorithm is parameterized with conventional propagators for the corresponding deterministic version of the global chance constraint. By using our novel modeling approach, we obtain significantly more compact model formulations than the state-of-the-art approach in [26].

We extended the generic filtering algorithm in two ways in order to obtain two incremental variations: a lightweight version (Algorithm 2) as well as a memory-intensive one (Algorithm 3). We performed an extensive experimental study on three benchmark problems: two stochastic constraint satisfaction problems and a stochastic constraint optimization one. This experimental study revealed that:

- by using the non-incremental filtering we outperform the state-of-the-art approach in [26] in terms of pruning and runtimes;
- the incremental versions of the filtering algorithm are computationally more efficient than the non-incremental filtering;
- Algorithms 2 and 3 are in principle incomparable with each other, since the efficiency of both is strictly influenced by the consistency level enforced by algorithm $\mathcal{A}$; and
- when the number of scenarios grows, the proposed approach is more scalable than the state-of-the-art approach in [26].

Future work may investigate opportunities offered by the integration of cost-based filtering techniques for solving stochastic constraint optimization problems such as the stochastic knapsack (see e.g. [25]). Another important future direction is the investigation of how sampling techniques may improve scalability and efficiency of our approach.

Appendix A

We discuss a filtering strategy for handling constraint expressions involving expected values (Section 8.3.1).

Algorithm 4: Filtering expected values.

input : $\langle \text{exp} \rangle$; $\mathcal{PT}$; x.
output: Bound consistent x.

```
1 begin
2     UB ← 0;
3     LB ← 0;
4     for each p ∈ Ψ do
5         UB ← UB + Sup(⟨exp⟩_↓p) · Pr{arcs(p)};
6         LB ← LB + Inf(⟨exp⟩_↓p) · Pr{arcs(p)};
7     Sup(x) ← UB;
8     Inf(x) ← LB;
9 end
```

Consider a constraint $x = \mathbb{E}[\langle \text{exp} \rangle]$, where x is a real valued decision variable, whose domain is stored as an interval with real valued upper and lower bounds. Techniques for handling propagation and search involving real valued decision variables are discussed in [2]. A filtering algorithm that enforces bounds consistency on this constraint is shown in Fig. 4. The algorithm simply evaluates two values: UB and LB. UB denotes an upper bound for the expected value of $\langle \text{exp} \rangle$, LB denotes a lower bound for the expected value of $\langle \text{exp} \rangle$. It should be noted that the algorithm operates by exploiting the structure Ψ of the policy tree. Therefore it takes implicitly into account the stage structure of the problem while computing the expected value of a given expression. For this reason, the algorithm will correctly evaluate expected values both in a single or multi-stage case. Furthermore, more complex objective functions can be easily implemented by incorporating the required expression $\langle \text{exp} \rangle$ — for instance $\max(\sum_{i=1}^{k} \omega_i x_i - c, 0)$ in the case of penalty costs for buying additional capacity — in the filtering strategy discussed in Fig. 4.

References

[1] T. Balafoutis, K. Stergiou, Algorithms for stochastic CSPs, in: F. Benhamou (Ed.), Principles and Practice of Constraint Programming – CP 2006, Proceedings of the 12th International Conference, CP 2006, Nantes, France, September 25–29, 2006, in: Lecture Notes in Computer Science, vol. 4204, Springer, Berlin/Heidelberg, 2006, pp. 44–58.

[2] F. Benhamou, L. Granvilliers, Continuous and interval constraints, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006 (Ch. 16).

[3] T. Benoist, E. Bourreau, Y. Caseau, B. Rottembourg, Towards stochastic constraint programming: A study of online multi-choice knapsack with deadlines, in: T. Walsh (Ed.), Principles and Practice of Constraint Programming – CP 2001, Proceedings of the 7th International Conference, CP 2001, Paphos, Cyprus, November 26–December 1, 2001, in: LNCS, vol. 2239, Springer, 2001, pp. 61–76.

[4] C. Bessiere, E. Hebrard, B. Hnich, T. Walsh, The complexity of reasoning with global constraints, Constraints 12 (2) (2007) 239–259.

[5] J.R. Birge, F. Louveaux, Introduction to Stochastic Programming, Springer Verlag, New York, 1997.

[6] L. Bordeaux, H. Samulowitz, On the stochastic constraint satisfaction framework, in: Y. Cho, R.L. Wainwright, H. Haddad, S.Y. Shin, Y.W. Koo (Eds.), Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11–15, 2007, ACM, 2007, pp. 316–320.

[7] A. Charnes, W.W. Cooper, Deterministic equivalents for optimizing and satisficing under chance constraints, Operations Research 11 (1) (1963) 18–39.

[8] H. Fargier, J. Lang, R. Martin-Clouaire, T. Schiex, A constraint satisfaction framework for decision under uncertainty, in: P. Besnard, S. Hanks (Eds.), UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, August 18–20, 1995, Montreal, Quebec, Canada, Morgan Kaufmann, 1995, pp. 167–174.

[9] M.R. Fellows, T. Friedrich, D. Hermelin, N. Narodytska, F.A. Rosamond, Constraint satisfaction problems: Convexity makes alldifferent constraints tractable, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16–22, 2011, IJCAI/AAAI, 2011, pp. 522–527.

[10] I.P. Gent, C. Jefferson, I. Miguel Minion, A fast scalable constraint solver, in: G. Brewka, S. Coradeschi, A. Perini, P. Traverso (Eds.), ECAI 2006, Proceedings of the 17th European Conference on Artificial Intelligence, August 29–September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), in: Frontiers in Artificial Intelligence and Applications, vol. 141, IOS Press, 2006, pp. 98–102.

[11] B. Hnich, R. Rossi, S.A. Tarim, S.D. Prestwich, Synthesizing filtering algorithms for global chance-constraints, in: I.P. Gent (Ed.), Principles and Practice of Constraint Programming – CP 2009, Proceedings of the 15th International Conference, CP 2009, Lisbon, Portugal, September 20–24, 2009, in: Lecture Notes in Computer Science, vol. 5732, Springer, 2009, pp. 439–453.

[12] B. Hnich, R. Rossi, S.A. Tarim, S.D. Prestwich, A survey on CP-AI-OR hybrids for decision making under uncertainty, in: M. Milano, P. Van Hentenryck (Eds.), Hybrid Optimization: The 10 Years of CP-AI-OR, in: Springer Optimization and Its Applications, vol. 45, Springer, 2011, pp. 227–270.

[13] H. Jeffreys, Theory of Probability, Clarendon Press, Oxford, UK, 1961.

[14] M. Kutz, K. Elbassioni, I. Katriel, M. Mahajan, Simultaneous matchings: Hardness and approximations, Journal of Computer and System Sciences 74 (5) (2008) 884–897.

[15] F. Laburthe, Choco: Implementing a CP kernel, in: N. Beldiceanu, W. Harvey, M. Henz, F. Laburthe, E. Monfroy, T. Müller, L. Perron, C. Schulte (Eds.), Proceedings of TRICS: Techniques for Implementing Constraint Programming Systems, a Post-Conference Workshop of CP 2000, Singapore, September 18–21, 2000, pp. 71–85.

[16] A. López-Ortiz, C.-G. Quimper, J. Tromp, P. van Beek, A fast and simple algorithm for bounds consistency of the alldifferent constraint, in: G. Gottlob, T. Walsh (Eds.), IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9–15, 2003, Morgan Kaufmann, 2003, pp. 245–250.

[17] R. McGill, J.W. Tukey, W.A. Larsen, Variations of box plots, The American Statistician 32 (1) (1978) 12–16.

[18] S.D. Prestwich, S.A. Tarim, R. Rossi, B. Hnich, Evolving parameterised policies for stochastic constraint programming, in: I.P. Gent (Ed.), Principles and Practice of Constraint Programming – CP 2009, Proceedings of the 15th International Conference, CP 2009, Lisbon, Portugal, September 20–24, 2009, in: Lecture Notes in Computer Science, vol. 5732, Springer, 2009, pp. 684–691.

[19] S.D. Prestwich, S.A. Tarim, R. Rossi, B. Hnich, Stochastic constraint programming by neuroevolution with filtering, in: A. Lodi, M. Milano, P. Toth (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Proceedings of the 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14–18, 2010, in: Lecture Notes in Computer Science, vol. 6140, Springer, 2010, pp. 282–286.

[20] J.-C. Régin, J.-F. Puget, A filtering algorithm for global sequencing constraints, in: G. Smolka (Ed.), Principles and Practice of Constraint Programming – CP97, Proceedings of the Third International Conference, Linz, Austria, October 29–November 1, 1997, in: Lecture Notes in Computer Science, vol. 1330, Springer, 1997, pp. 32–46.
[21] R. Rossi, B. Hnich, S.A. Tarim, S. Prestwich, Finding (α, ϑ)-solutions via sampled SCSP, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16–22, 2011, IJCAI/AAAI, 2011, pp. 2172–2177.
[22] R. Rossi, S.A. Tarim, R. Bollapragada, Constraint-based local search for computing non-stationary replenishment cycle policy under stochastic lead-times, INFORMS Journal on Computing 24 (1) (2012) 66–80.
[23] R. Rossi, S.A. Tarim, B. Hnich, S. Prestwich, A global chance-constraint for stochastic inventory systems under service level constraints, Constraints 13 (4) (2008) 490–517.
[24] R. Rossi, S.A. Tarim, B. Hnich, S. Prestwich, Computing replenishment cycle policy under non-stationary stochastic lead time, International Journal of Production Economics 127 (1) (2010) 180–189.
[25] R. Rossi, S.A. Tarim, B. Hnich, S.D. Prestwich, Cost-based domain filtering for stochastic constraint programming, in: P.J. Stuckey (Ed.), Principles and Practice of Constraint Programming, Proceedings of the 14th International Conference, CP 2008, Sydney, Australia, September 14–18, 2008, in: Lecture Notes in Computer Science, vol. 5202, Springer, 2008, pp. 235–250.
[26] S.A. Tarim, S. Manandhar, T. Walsh, Stochastic constraint programming: A scenario-based approach, Constraints 11 (1) (2006) 53–80.
[27] J.W. Tukey, Exploratory Data Analysis, 1st edition, Addison Wesley, 1977.
[28] T. Walsh, Stochastic constraint programming, in: F. van Harmelen (Ed.), Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002, IOS Press, 2002, pp. 111–115.