Synthesizing Filtering Algorithms for Global Chance-Constraints*

Brahim Hnich[1], Roberto Rossi[2], S. Armagan Tarim[3] and Steven Prestwich[4]

Faculty of Computer Science, Izmir University of Economics, Turkey[1]
brahim.hnich@ieu.edu.tr
Logistics, Decision and Information Sciences, Wageningen UR, the Netherlands[2]
roberto.rossi@wur.nl
Operations Management Division, Nottingham University Business School, UK[3]
armtar@yahoo.com
Cork Constraint Computation Centre, University College Cork, Ireland[4]
s.prestwich@4c.ucc.ie

Abstract. Stochastic Constraint Satisfaction Problems (SCSPs) are a powerful modeling framework for problems under uncertainty. To solve them is a P-Space task. The only solution approach to date compiles down SCSPs 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 some of 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 enhances constraint propagation. Experiments show the benefits of this novel approach.

1 Introduction

Stochastic Constraint Satisfaction Problems (SCSPs) are a powerful modeling framework for problems under uncertainty. SCSPs were first introduced in [10] and further extended in [9] to permit multiple chance-constraints and a range of different objectives in order to model combinatorial problems under uncertainty. SCSP is a PSPACE-complete problem [10]. The approach in [9] compiles down SCSPs into deterministic equivalent CSPs. This makes it possible to reuse existing solvers, but at the cost of increased space requirements and of hindering constraint propagation. In this paper we overcome some of 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 [9], our approach reuses the propagators already available for classical CSPs. But, unlike [9], our approach

* Brahim Hnich is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. SOBAG-108K027.

uses fewer decision variables and strengthens constraint propagation. Our results show that our approach is superior to the one in [9], since it achieves stronger pruning and therefore it proves to be more efficient in terms of run time and explored nodes.

The paper is structured as follows: in Section 2 we provide the relevant formal background; in Section 3 we discuss the structure of a SCSP solution; in Section 4 we describe the state-of-the-art approach to SCSPs; in Section 5 we discuss our novel approach; in Section 6 we present our computational experience; in Section 7 we provide a brief literature review; finally, in Section 8 we draw conclusions.

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) iff 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* [9]. 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 [5] 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 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

$$\mathcal{N} = 1 + |s_1| + |s_1| \cdot |s_2| + \ldots + |s_1| \cdot |s_2| \cdot \ldots \cdot |s_{m-1}|$$

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 $\langle x_2, \bar{x}_2, |s_1| + |s_1|.|s_2|\rangle$, 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. $p = [\langle x_1, \bar{x}_1, 0\rangle, \langle s_1, \bar{s}_1\rangle, \ldots, \langle x_m, \bar{x}_m, k\rangle, \langle s_m, \bar{s}_m\rangle]$. 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 $nodes(p) = [\langle x_1, \bar{x}_1, 0\rangle, \ldots, \langle x_m, \bar{x}_m, j\rangle]$. 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

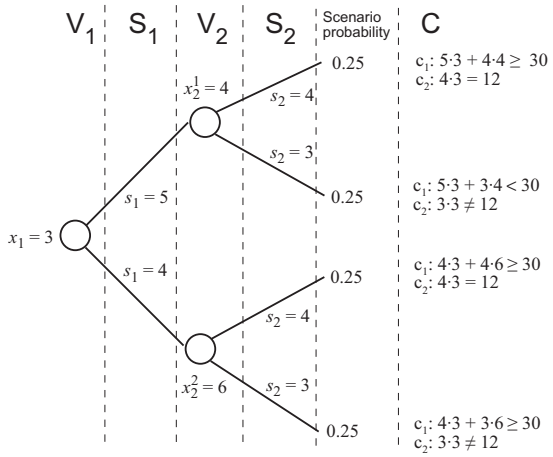

Fig. 1. Policy tree for the SCSP in Example 1

$nodes(p)$. We say that h is *satisfied wrt to a given policy tree* $\mathcal{T}$ iff

$$\sum_{p \in \Psi : h_{\downarrow p} \in h_{\downarrow p}} \Pr\{arcs(p)\} \geq \theta_h.$$

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 wrt* $\mathcal{T}$*.*

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[1] in C, $c_1 : \Pr\{s_1 x_1 + s_2 x_2 \geq 30\} \geq 0.75$ and $c_2 : \Pr\{s_2 x_1 = 12\} \geq 0.5$. 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:
$p_1 = [\langle x_1, 3, 0\rangle, \langle s_1, 5\rangle, \langle x_2, 4, 1\rangle, \langle s_2, 4\rangle]$, $p_2 = [\langle x_1, 3, 0\rangle, \langle s_1, 5\rangle, \langle x_2, 4, 1\rangle, \langle s_2, 3\rangle]$,
$p_3 = [\langle x_1, 3, 0\rangle, \langle s_1, 4\rangle, \langle x_2, 6, 2\rangle, \langle s_2, 4\rangle]$, $p_4 = [\langle x_1, 3, 0\rangle, \langle s_1, 4\rangle, \langle x_2, 6, 2\rangle, \langle s_2, 3\rangle]$.

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

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

4 Scenario-based Approach to Solve SCSPs

In [9], the authors discuss an equivalent scenario-based reformulation for SCSPs. 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.

Constraints:

(1) $(5x_1^1 + 4x_2^1 \geq 30) \leftrightarrow (Z_{c_1}^1 = 1)$
(2) $(5x_1^1 + 3x_2^1 \geq 30) \leftrightarrow (Z_{c_1}^2 = 1)$
(3) $(4x_1^1 + 4x_2^2 \geq 30) \leftrightarrow (Z_{c_1}^3 = 1)$
(4) $(4x_1^1 + 3x_2^2 \geq 30) \leftrightarrow (Z_{c_1}^4 = 1)$
(5) $\sum_{\omega=1}^{4} 0.25 Z_{c_1}^{\omega} \geq \theta_{c_1}$
(6) $(4x_1^1 = 12) \leftrightarrow (Z_{c_2}^1 = 1)$
(7) $(3x_1^1 = 12) \leftrightarrow (Z_{c_2}^2 = 1)$
(8) $(4x_1^1 = 12) \leftrightarrow (Z_{c_2}^3 = 1)$
(9) $(3x_1^1 = 12) \leftrightarrow (Z_{c_2}^4 = 1)$
(10) $\sum_{\omega=1}^{4} 0.25 Z_{c_2}^{\omega} \geq \theta_{c_2}$

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

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),...,(5), whilst chance-constraint c_2 is compiled down into constraints (6),...,(10). Constraints (1),...,(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 $\bar{s}_1 x_1^1 + \bar{s}_2 x_2^i \geq 30$ — 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 or equal to the required threshold $\theta_{c_1} = 0.75$. A similar reasoning applies to constraints (6),...,(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. These drawbacks are:

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

Hindering Constraint Propagation: the holistic CSP heavily depends on reification constraints for constraint propagation, which is a very weak form of propagation. Also, if the chance-constraint involves a global constraint (e.g., $\Pr\{\text{alldiff}(x_1, s_1, x_2)\} \geq \theta$), 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 any of the current constraint solvers.

5 Generic Filtering Algorithms

In this section we show how to overcome the drawbacks discussed above.

5.1 Theoretical Properties

Like the approach in [10], 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:

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

This array of decision variables is shared among the constraints in the model similarly to what happens with decision variables in classic CSPs. In what follows, we will discuss how to propagate chance-constraints on the policy tree decision variable array.

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 wrt** *h iff there exists an assignment of values to variables in $\mathcal{PT}$ that is a satisfying policy wrt 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 wrt h.*

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

For convenience, given a chance-constraint $h \in C$, we now redefine $h_{\downarrow p}$ as the resulting deterministic constraint in which we substitute 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 — according to our former definition — in which we substitute every stochastic variable s_i with the corresponding values ($\bar{s}_i$) 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. We denote by $h^{i,v}_{\downarrow p}$ the 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. $h_{\downarrow p}$. Let $\Psi_i = \{p \in \Psi | h_{\downarrow p}$ *constrains* $\mathcal{PT}[i]\}$. We introduce $f(i, v)$ as follows:

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

where $f(i, v)$ is the sum of the probabilities of the scenarios in which value v in the domain of $\mathcal{PT}[i]$ is consistent. As the next proposition shows, one can exploit this 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 wrt h; where $\max(j) = \max\{f(j, v) | v \in D(\mathcal{PT}[j])\}$.

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. □

5.2 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.

Case 1: Algorithm 1 takes as input chance-constraint h, $\mathcal{PT}$, and a propagator $\mathcal{A}$. It filters from $\mathcal{PT}$ inconsistent values wrt h. For each decision variable and each value in its domain, we initialize $f[i, v]$ to 0 in line 2. In 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 it constrains. Then we enforce GAC on c using $\mathcal{A}$. We iterate through the domain of each copy of the decision variable at index i 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. In 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 $max[i]$. In 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 $f[i, v]$.

Algorithm 1: `Filtering Algorithm`

input : h; $\mathcal{PT}$; $\mathcal{A}$.
output: Filtered $\mathcal{PT}$ wrt 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
```

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

Proof: **(Sketch) Soundness.** When a value v is pruned by Algorithm 1, Proposition 1 is 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. 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 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. 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 $f[1, v] = 0.25$ 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

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

Table 1. Example of inconsistent values gone undetected

reach a fixed-point and no further pruning is done. We denote as $\mathcal{H}$ this modified algorithm.

Theorem 2 *Algorithm $\mathcal{H}$ 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.
(Sketch) 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 $\mathcal{H}$. The values that are not pruned when $\theta = 0.75$ are underlined. Note that if we propagate the constraints in the model generated according to the approach described in [9] and shown in Fig. 2, no value is pruned.

Even though algorithm $\mathcal{H}$ is a sound filtering algorithm, it is unfortunately still incomplete.

$\mathcal{PT}[0]$	$f[0,v]$	$\mathcal{PT}[1]$	$f[1,v]$	$\mathcal{PT}[2]$	$f[2,v]$
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 2. Pruning for Example 2 after calling Algorithm $\mathcal{H}$

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

Table 3. Filtered domains

Theorem 3 *The level of consistency achieved by algorithm $\mathcal{H}$ on global chance-constraint h is weaker than GAC on h.*

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

Indeed, we conjecture that maintaining GAC on a global chance-constraint is intractable in general even if maintaining GAC on its deterministic version is polynomial.

Case 2: When $\theta_h = 1$ the global chance-constraint h can be reformulated as $h_{\downarrow p}, \forall p \in \Psi$. If all deterministic constraints are simultaneously GAC, then this reformulation is equivalent to algorithm $\mathcal{H}$. Nevertheless, even in this special case, we still lose in terms of pruning.

Theorem 4 *GAC on h is stronger than GAC on the reformulation.*

Proof: 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. □

6 Computational Experience

In this section, we present our computational experience, which shows that our approach outperforms the state-of-the-art approach in [9] both in terms of run time and explored nodes, and that it is also able to achieve stronger pruning.

In our experiments we considered a number of randomly generated SCSPs. The SCSPs considered feature five chance-constraints over 4 integer decision variables, $x_1, \ldots, x_4$ and 8 stochastic variables, $s_1, \ldots, s_8$. 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 $\frac{1}{3}$. There are five chance-constraints in the model, the first embeds an equality, $c_1 : \Pr\{x_1s_1 + x_2s_3 + x_3s_5 + x_4s_7 = 80\} \geq \alpha$, the second and the third embed inequalities, $c_2 : \Pr\{x_1s_2 + x_2s_4 + x_3s_6 + x_4s_8 \leq 100\} \geq \beta$ and $c_3 : \Pr\{x_1s_2 + x_2s_4 + x_3s_6 + x_4s_8 \geq 60\} \geq \beta$. 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. 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: $c_4 : \Pr\{x_1s_2 + x_3s_6 \geq 30\} \geq 0.7$. Finally, the fifth chance-constraint embeds an equality also defined over a subset of all the decision and random variables in the model: $c_5 : \Pr\{x_2s_4 + x_4s_8 = 20\} \geq 0.05$.

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 $V_4 = \{x_4\}$, $S_4 = \{s_4, s_8\}$.

The propagation strategy discussed in Section 5 requires an existing propagator $\mathcal{A}$ for the deterministic constraints. Since the only constraints appearing in the SCSPs described above are linear (in)equalities, we borrowed a simple bound-propagation procedure for linear (in)equalities implemented in Choco 1.2 [6], a JAVA open source CP solver. 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.

In order to assess efficiency and effectiveness, 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 [9] (SBA).

Firstly, we wish to underline that SBA, the approach discussed in [9], requires a much larger number of constraints and decision variables to model the prob-

lems above. Specifically, the single-stage problem is modeled, in [9], using 6484 decision variables and 6485 constraints, while GCC — our approach — requires only 4 decision variables and 5 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 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 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 constraints.

As discussed above, in our comparative study we considered 18 different possible configurations for the parameters α and β. For each of these configurations, we generated 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. To each instance we assigned a time limit of 240 seconds for running the search. The computational performances of GCC and SBA are compared in Fig. 3. All the experiments were performed on an Intel(R) Centrino(TM) CPU 1.50GHz with 2Gb RAM. The solver used for our test is Choco 1.2 [6].

In the test bed considered, GCC solved, in the given time limit of 240 seconds, all the instances that SBA could solve within this time limit. In contrast, SBA was often not able to solve — within the given time limit of 240 secs — instances that GCC could solve in a few seconds. More specifically, both GCC and SBA could solve 90 over 90 1-stage instances; on average GCC explored roughly 5 times less nodes and was about 2.5 times faster than SBA for these instances. GCC could solve 45 over 90 2-stage instances, while SBA could only solve 18 of them; on average GCC explored roughly 400 times less nodes and was about 13 times faster than SBA for these instances. Finally, GCC could solve 31 over 90 4-stage instances, while SBA could only solve 10 of them; on average GCC explored roughly 300 times less nodes and was about 15 times faster than SBA for these instances.

In our computational experience, we also compared the effectiveness of the filtering performed by SBA and GCC. In order to do so, we considered 90 two-stage feasible instances randomly generated according to the strategy discussed above (5 different probability distributions for the random variables and 18 different configurations for the parameters α and β). We considered a solution for each of these instances, we randomly picked subsets of the decision variables in the problem, we assigned them to the value they take in this solution, we propagated according to SBA and GCC, respectively, and we compared the percentage of values pruned by each of these two approaches. In Fig. 4 we show the results of this comparison, which is performed for a number of decision variables

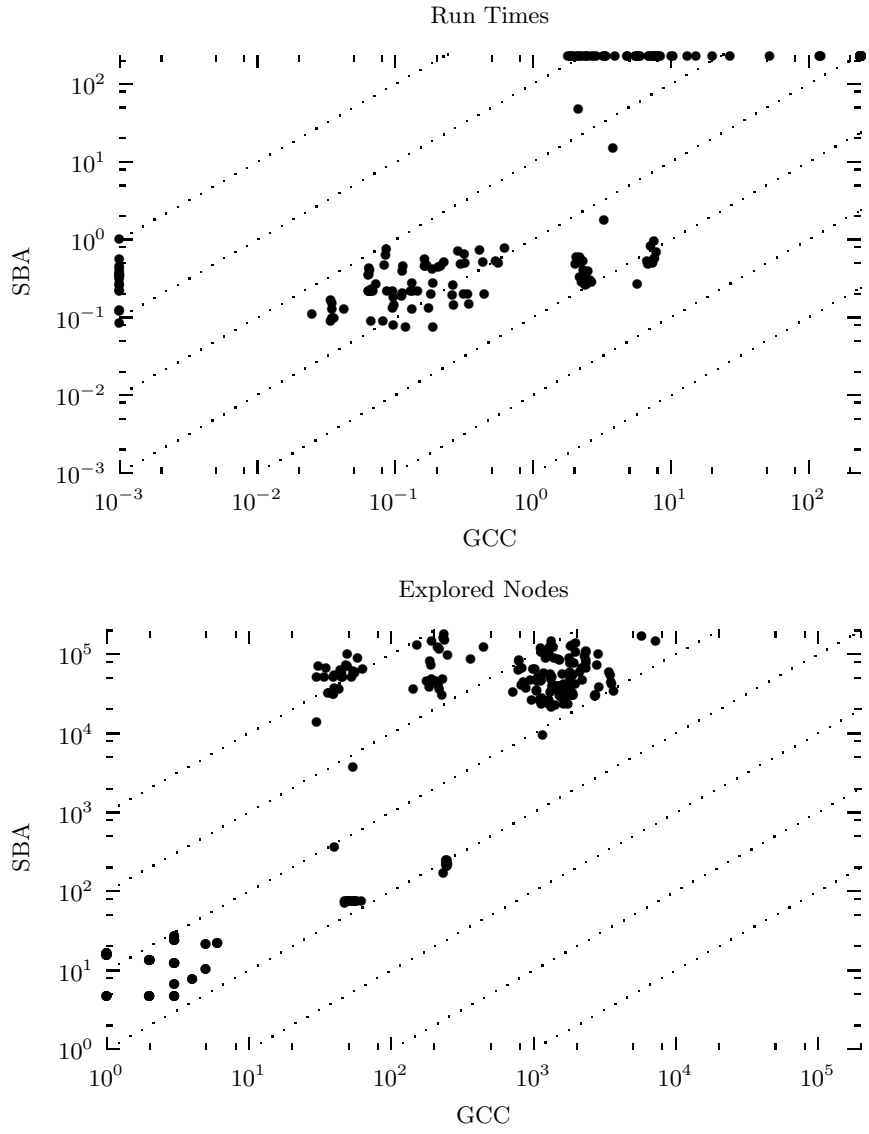

Fig. 3. Scatter graphs for our computational experience. The top graph compares the run time performance of SBA and GCC for the 270 instances in our test bed. The bottom graph shows, instead, a comparison in terms of explored nodes.

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

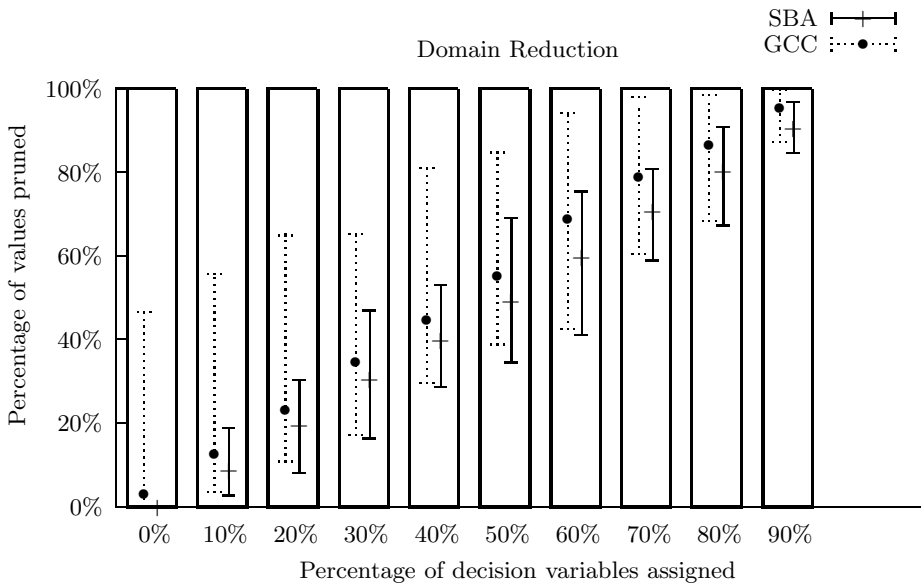

Fig. 4. Effectiveness of the filtering performed by SBA and GCC

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.

7 Related Works

Closely related to our approach are [7, 8]. 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 [10] 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 a SCSP. Finally, efforts that try to extend classical CSP framework to incorporate uncertainty have been influenced by works that originated in different fields, namely *chance-constrained programming* [4] and *stochastic programming* [3]. 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. [2]. The idea of employing a scenario-based approach for building up constraint programming models of SCSPs is not novel, since Tarim et al. [9] have already used this technique to develop a fully featured language — Stochastic OPL — for modeling SCSPs. Our work proposes an orthogonal approach to solving SCSPs that could easily be integrated with the compilation approach of [9] to make it more efficient.

8 Conclusions

We proposed generic filtering algorithms for global chance-constraints. Our filtering algorithms are parameterized with conventional propagators for the corresponding deterministic version of the global chance-constraint. Our experimental results show that our approach outperforms the approach in [9], both in terms of run time and explored nodes. We also showed, experimentally, that our approach produces stronger pruning than the approach in [9]. An interesting open question is to determine if it is tractable to maintain GAC on global chance-constraints for which GAC on the corresponding deterministic constraints is tractable. Future works may investigate the tractability of GAC for classes of global chance-constraints and ways of making algorithm $\mathcal{H}$ incremental.

References

1. T. Balafoutis and K. Stergiou. Algorithms for stochastic csps. In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming, CP 2006, Proceedings*, volume 4204 of *LNCS*, pages 44–58. Springer, 2006.
2. T. Benoist, E. Bourreau, Y. Caseau, and B. Rottembourg. Towards stochastic constraint programming: A study of online multi-choice knapsack with deadlines. In Toby Walsh, editor, *Principles and Practice of Constraint Programming, CP 2001, Proceedings*, volume 2239 of *LNCS*, pages 61–76. Springer, 2001.
3. J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, New York, 1997.
4. A. Charnes and W. W. Cooper. Deterministic equivalents for optimizing and satisficing under chance constraints. *Operations Research*, 11(1):18–39, 1963.
5. H. Jeffreys. *Theory of Probability*. Clarendon Press, Oxford, UK, 1961.
6. F. Laburthe and the OCRE project team. Choco: Implementing a cp kernel. Technical report, Bouygues e-Lab, France, 1994.
7. R. Rossi, S. A. Tarim, B. Hnich, and S. Prestwich. A global chance-constraint for stochastic inventory systems under service level constraints. *Constraints*, 13(4):490–517, 2008.
8. R. Rossi, S. A. Tarim, B. Hnich, and S. D. Prestwich. Cost-based domain filtering for stochastic constraint programming. In Peter J. Stuckey, editor, *Principles and Practice of Constraint Programming, CP 2008, Proceedings*, volume 5202 of *LNCS*, pages 235–250. Springer, 2008.
9. S. A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.
10. T. Walsh. Stochastic constraint programming. In Frank van Harmelen, editor, *European Conference on Artificial Intelligence, ECAI'2002, Proceedings*, pages 111–115. IOS Press, 2002.