# Boosting Probabilistic Choice Operators

Matthieu Petit* and Arnaud Gotlieb

IRISA – INRIA, Campus Beaulieu, 35042 Rennes Cedex, France
{Matthieu.Petit,Arnaud.Gotlieb}@irisa.fr

**Abstract.** Probabilistic Choice Operators (PCOs) are convenient tools to model uncertainty in CP. They are useful to implement randomized algorithms and stochastic processes in the concurrent constraint framework. Their implementation is based on the random selection of a value inside a finite domain according to a given probability distribution. Unfortunately, the probabilistic choice of a PCO is usually delayed until the probability distribution is completely known. This is inefficient and penalizes their broader adoption in real-world applications. In this paper, we associate to PCO a filtering algorithm that prunes the variation domain of its random variable during constraint propagation. Our algorithm runs in $O(n)$ where $n$ denotes the size of the domain of the probabilistic choice. Experimental results show the practical interest of this approach.

## 1 Introduction

**Motivations.** Reasoning with uncertainty is an important issue in Constraint Programming as many real-world problems (resource management, network traffic analysis, energy trading) invariably include incomplete or unknown parameters. Besides the original probabilistic extension of Constraint Satisfaction Problems proposed by Fargier and Lang [4], these last years have seen the development of several frameworks that offer capabilities to model uncertainty with probabilities. In 2000, Walsh proposed Stochastic Constraint Programming that extends CP by including both decision variables, that can be set, and stochastic variables that follow a given probability distribution [16,14]. In the concurrent constraint framework, Di Pierro and Wiklicky [2] and Gupta, Jagadeesan and Saraswat [6] introduced Probabilistic Choice Operators (PCOs) in concurrent constraint processes to implement randomized algorithms [3] and stochastic processes [5]. In all these frameworks, random draws have to be performed over completely known probability distributions. But, as noted by Yorke-Smith and Gervet [17], for many real-world problems, exact probability distributions are unknown or just implicitly defined via some additional constraints. For example, in resources assignment problems, one often knows that the probability to sell an item during summer time is five times lower than the probability to sell it during winter but we don't know the exact probability distribution to sell it over the year. In some

* This work is part of the GENETTA project granted by the Brittany region.

problems, the probability distribution itself is what we are looking for. Typical examples include biased games models, hardware and software constraint-based verification. For example, consider the Statistical Structural Testing technique [15] in Software Engineering. The problem aims at finding a distribution probability over the input domain of a program, that maximizes the coverage of some structural criteria, such as all_statements or all_paths. Up to know, there is no satisfactory automated technique for biasing the choice of test data that meet this objective. It has been shown in [9] that this problem can be modelled as a stochastic constraint solving problem for which the solution is the probability distribution. In order to model accurately such problems, probabilistic choices are currently delayed until complete information over the probability distribution is available. In terms of constraint modelling, this corresponds to enumerate the possible probability distributions and to solve each of the corresponding stochastic constraint system. This is inefficent and penalizes the broader adoption of PCOs in real-world applications.

**Contributions.** In this paper, we associate to PCOs a filtering algorithm that prunes the domain of the stochastic variable of the probabilistic choice. Our purpose is to boost the probabilistic decision even if the probability distribution is unknown or just known via some additional constraints. The main idea is to benefit from the early random draw of an auxiliary stochastic variable in order to prune the variation domain of the probability distribution during constraint propagation. Thanks to the availability of this random value, our filtering algorithm can eliminate the values of the probabilistic choice that are incompatible with current state of the PCO. Interestingly, this algorithm runs in $O(n)$ where $n$ denotes the size of the domain of the probabilistic choice.

As a basic example, consider the stochastic optimization problem of Fig.1 inspired from a production planning problem. The problem aims at finding a value for $X$ that minimizes $X - Y$ where $Y$ is a stochastic variable, the value of which is given by the random draw of a biased die. Suppose we ignore the exact bias of the die and we just know that the 6-face of the die is two times more overloaded than the 1-face, the 5-face is two times more overloaded than the 2-face, and the 4-face is two times more overloaded than the 3-face. The PCO `choose` picks up at random a value of the die according to the unknown distribution probability $[1, 2, 3, 4, 5, 6] - [W1, W2, W3, W4, W5, W6]$ where $Wi$ denotes the weight associated to face $i$ of the die. The simplest approach to solve this problem consists in labeling first on the possible distribution probabilities in order to allow the random draw of $Y$ (option `no_filtering` in Fig.1). Unfortunately, this approach rapidely becomes intractable. `R` denotes the mean CPU time (in sec.) obtained by launching 2000 times the solving process. When our filtering algoithm is used (option `domain_bound`), the possible probability distributions are pruned before labelling. On this example, our gain is more than one order of magnitude in average.

We implemented our filtering algorithm in several PCOs of the SICStus Prolog library PCC(FD). Experimental results show the practical interest of our approach.

```
s_optim(Opt) :-
        X in 1..6,
        domain([W1,W2,W3,W4,W5,W6], 1, 10),
        choose(Y,[1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt), % PCO call
        X #>= Y,
        2*W1 #= W6, % 6-face is two times more overloaded than 1-face
        2*W2 #= W5,
        2*W3 #= W4,
        labeling([minimize(X-Y)],[X,W1,W2,W3,W4,W5,W6]).

?- bench([s_optim([no_filtering]), 2000, R).
R = 9.479 ?

?- bench([s_optim([domain_bound]), 2000, R). % filtering on choose
R = 0.646 ?
```

**Fig. 1.** A stochastic optimization problem on SICStus 3.11, Intel-Pentium 2Ghz, 1Go RAM, WinXP

**Organization.** The paper is organized as follows : Section 2 briefly describes the theoretical background on PCOs required to understand the rest of the paper. Section 3 presents the principles of filtering associated to PCO. Section 4 describes our implementation of several PCOs in SICStus Prolog, while section 5 presents the experimental results. Finally, Section 6 indicates several perspectives to this work.

## 2 Background on Probabilistic Choice Operators

Probabilistic Choice Operators have been introduced originally within the Concurrent Constraint framework of Saraswat. In the first subsection, we start by briefly recalling the principles of the theoretical scheme: Probabilistic Concurrent Constraint Programming (PCCP). In the second subsection, we focus on Finite Domains which form a computational domain of the generic scheme PCCP.

### 2.1 Probabilistic Concurrent Constraint Programming

**Concurrent Constraint Programming (CCP).**
We start by recalling some syntax and semantics elements of CCP. In CCP, processes are executed concurrently and can interact with each other through a common constraint store. A CCP language is parameterized by a constraint system [12], which is composed of a set of primitive constraints and an entailment relation. The syntax of a CCP language is given by the following grammar:

$$Process ::= tell(C) \,|\, if\ C\ then\ Process \,|\, new\ X\ in\ Process \,|\, Process \| Process.$$

where $tell(C)$ adds the constraint $C$ to the constraint store, $if\ C\ then\ Process$ asks whether $C$ is entailed by the current constraint store and adds the constraints of $Process$ if $C$ is entailed, $new\ X\ in\ Process$ adds the constraints of $Process$ to the store while hiding the variable $X$ from other processes, and finally, $\|$ represents the parallel composition that can be interpreted as a logical conjunction in a Logic Programming environment. Well known examples of CCP languages include cc(FD)[7], AKL[8] or Oz/Mozart[13] just to name a few.

**Probabilistic Choice Operators.**
A few years ago, Gupta et al. [6,5] and Di Pierro and Wiklicky [2,3] proposed to add probabilistic choice operators to CCP. In our presentation, we will confine ourselves to the PCO *choose* [6]. Formally, the operator $choose(X, Law_X, Process)$ injects a stochastic variable $X$ along with a probabilistic law $Law_X$ into a concurrent process $Process$. For the sake of simplicity, we suppose that $Law_X$ takes the form of a pair $[v_1, \ldots, v_n] - [w_1, \ldots, w_n]$ where $[v_1, \ldots, v_n]$ denotes the list of possible (distinct) values for $X$, called the *domain* of the probabilistic choice, while $[w_1, \ldots, w_n]$ denotes the list of non-negative weights associated to the $v_i$, called the *probability distribution* of the probabilistic choice. In the rest of the paper, $dom(X) = [v_1, \ldots, v_n]$ denotes a sorted finite set of integer values associated to variable $X$. $min(X)$ (resp. $max(X)$) denotes the minimum (resp. maximum) of $dom(X)$.

Operationally, $choose(X, [v_1, \ldots, v_n] - [w_1, \ldots, w_n], Process)$ executes $Process_{X \leftarrow v_i}$ with a probability $p_i$. $Process_{X \leftarrow v_i}$ denotes the concurrent process $P$ where $X$ has been substituted by $v_i$ and $p_i$ denotes the probability of the event $X = v_i$ which is computed by the following formula:

$$p_i = \frac{w_i}{\sum_{j=1}^{n} w_j}.$$

Just to make things more concrete, we illustrate the processing of a PCCP request on a basic example extracted from [5]:

*Example 1.*

$$\begin{aligned} P = {} & choose(X, [0,1] - [1,1], tell(X = Z)) \parallel \\ & choose(Y, [0,1] - [1,1], if\ Z = 1\ then\ tell(Y = 1)). \end{aligned}$$

Roughly speaking, three possible terminal configurations can be obtained: $Z$ is constrained to 0 with the probability $\frac{1}{2}$ (event $X = 0$), $Z$ is constrained to 1 with the probability $\frac{1}{4}$ (event $X = 1 \wedge Y = 1$) and $false$ is obtained with the probability $\frac{1}{4}$ (event $X = 1 \wedge Y = 0$).

### 2.2 Probabilistic Choice Operator over Finite Domains

PCCP is parameterized by a computational domain over which the constraints are interpreted. In this paper, we focus on Finite Domains and we provide an operational semantics for PCO onto Finite Domains.

**Uncertainty on the probabilistic choice.**
The simulation (random draw) of values for the stochastic variable $X$ in a PCO is always possible when $Law_X$ is fully instantiated. On the contrary, when there is some uncertain data within the probabilistic choice, the simulation is delayed until all the parameters (domain, probability distribution) become instantiated. The set of uncertain data is characterized by the following definition.

**Definition 1.** *Let $X$ be a stochastic variable, let $Law_X = [v_1, \ldots, v_n] - [W_1, \ldots, W_n]$ be its probabilistic choice where $W_1, \ldots, W_n$ are Finite Domains variables, then the set of the possible probability distributions, called $\mathcal{S}L_X$ associated to $X$ is defined as follows:*

$$\mathcal{S}L_X \triangleq \big\{[v, \ldots, v_n] - [w_1, \ldots, w_n] \big| w_1 \in dom(W_1), \ldots, w_n \in dom(W_n)\big\}$$

This definition will be useful when we will introduce our filtering algorithm to prune the domain of the stochastic variable $X$. The filtering algorithm will be illustrated on the following example.

*Example 2.* Consider the example of a biased die where the 6-face is two times overloaded than the 1-face.

$$W_6 = 2 * W_1 \wedge choose(X, [1,2,3,4,5,6] - [W_1, W_2, W_3, W_4, W_5, W_6], X = Die)$$

Suppose that $dom(W_1) = 1..2$, $dom(W_2) = 2..2$, $dom(W_3) = 2..2$, $dom(W_4) = 2..2$, $dom(W_5) = 2..2$ and $dom(W_6) = 2..4$, then uncertainty on the unknown bias of the die is given by the following set:

$$\begin{aligned}\mathcal{SL}_X = \{\ & [1,2,3,4,5,6] - [1,2,2,2,2,2], [1,2,3,4,5,6] - [1,2,2,2,2,3], \\ & [1,2,3,4,5,6] - [1,2,2,2,2,4], [1,2,3,4,5,6] - [2,2,2,2,2,2], \\ & [1,2,3,4,5,6] - [2,2,2,2,2,3], [1,2,3,4,5,6] - [2,2,2,2,2,4]\}.\end{aligned}$$

**PCOs as constraint combinators.**
As said above, in the presence of uncertainty, the simulation of PCO should be delayed. Then, we considered PCOs as constraint combinators. This allows to reason on probabilistic choices that are only partially known and opens the door to apply constraint reasoning on PCOs. In this view, the probability distribution associated to a PCO is just constrained by a set of constraints on its possible values.

Base on that, constraint propagation over the *choose* operator can be defined. When *choose* is equipped with a filtering algorithm, one speak of the constraint combinator *choose*. $choose(X, Law_X, Process)$ succeeds whenever $X$ is valuated and $Process$ succeeds. When the probabilistic choice is only partially instantiated, *choose* is introduced into the propagation queue of the constraint solver and then, a filtering algorithm that is detailed below is launched to prune the domain of values for $X$. When no more pruning can be performed, the combinator falls asleep. It is awoken whenever the domain of at least one variable of the distribution probability is modified and then the combinator is reintroduced in the propagation queue. This process iterates until a fix point is reached, i.e. a state where no more deduction on the domain of $X$ is obtained.

## 3 Principle of the filtering on *choose*

In this section, we introduce the principles of the filtering on the *choose* combinator. This filtering algorithm permits to prune early the domain of the probabilistic choice. The main idea is to benefit from the early random draw of an auxiliary variable, the value of which is exploited within the filtering algorithm. First subsection is devoted to recall the principle of simulating random choice over Finite Domains. This principle is the basis of our filtering algorithm. Second subsection presents the correction properties on the *choose* combinator, while the third subsection presents the filtering algorithm. In this subsection, we prove correctness and termination of our algorithm and also give its complexity.

### 3.1 Simulation of a stochastic variable over Finite Domains

Stochastic variables over Finite Domains are usually simulated with the values of a uniform stochastic variable over $[0;1]$. This uniform stochastic variable is noted $U$. Our filtering algorithm for $choose(X, Law_X, Process)$ exploits an *a priori* random value of $U$ to prune $dom(X)$.

**Definition 2 (Distribution function).** *Let $X$ be a stochastic variable, let $Law_X = [v_1, \ldots, v_n] - [w_1, \ldots, w_n]$ be its probabilistic choice and $U$ an uniform stochastic variable over $[0;1]$, then a distribution function is a function $f$ that maps a random value of $U$ to a value of $X$, as follows:*

$$f : [0;1] \rightarrow Dom(X)$$
$$u \quad \mapsto \begin{cases} v_1 & \text{if } u \in \left[0, \frac{w_1}{\sum_{i=1}^{n} w_i}\right[ \\ \vdots & \vdots \\ v_n & \text{if } u \in \left[\frac{\sum_{i=1}^{n-1} w_i}{\sum_{i=1}^{n} w_i}, 1\right[ \end{cases}.$$

It is trivial to see that the probability of the event $X = v_i$ is equal to $P(u \in [pr_1 + \ldots + pr_{i-1}, pr_1 + \ldots + pr_i]) = (pr_1 + \ldots + pr_i) - (pr_1 + \ldots + pr_{i-1}) = pr_i$, as expected.

For example, consider the distribution function $f$ associated to a non-biased die, given by FIG. 2. Here, the distribution probability is given by $[1,2,3,4,5,6] - [1,1,1,1,1,1]$ and $u = 0.6$. As a result, $X = f(0.6) = 4$.

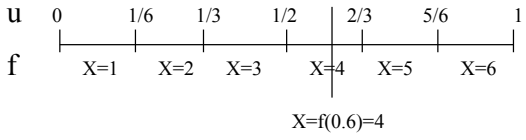


**Fig. 2.** Simulation of the stochastic variable $X$ with a uniform probability distribution on $\{1,2,3,4,5,6\}$

Associated to each probability distribution of $\mathcal{SL}_X$, a distribution function $f$ is defined. The set of distribution functions associated to $\mathcal{SL}_X$ is noted $\mathcal{F}_X$. For example, the set $\mathcal{F}_X$ of Example 2 is represented by FIG.3.

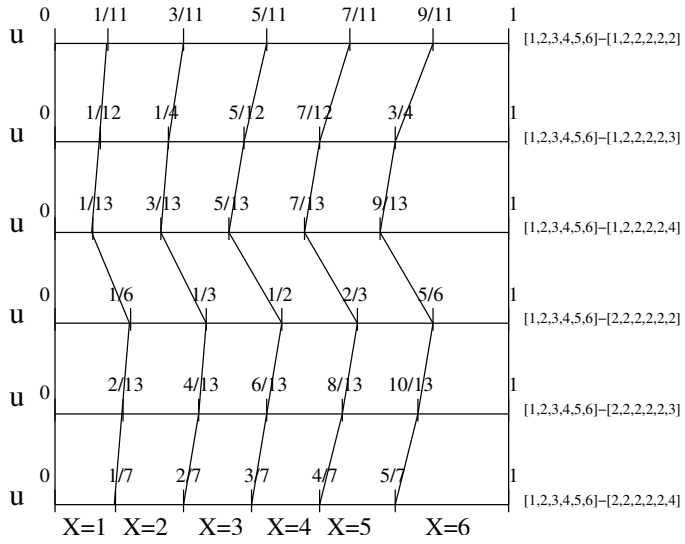


**Fig. 3.** Set of the distribution functions of the example 2 die

### 3.2 Stochastic consistencies on the *choose* combinator

We propose to characterize the consistency level achieved by our filtering combinator by using the notion of stochastic consistency. Stochastic consistency is a local consistency parameterized by an uniform random value $u$. In our framework, we define two stochastic consistencies on the $choose(X, Law_X, Process)$ combinator.

**Definition 3.** *(Stochastic domain consistency)*
*Let $u$ be a uniform random value and $\mathcal{F}_X$ be the set of distribution functions associated to $Law_X$, then the combinator $choose(X, Law_X, Process)$ is stochastic domain consistent iff*

$$\forall v_i \in dom(X),\ \exists f \in \mathcal{F}_X \text{ such that } f(u) = v_i$$

**Definition 4.** *(Stochastic bound consistency)*
*Let $u$ be a uniform random value and $\mathcal{F}_X$ be the set of distribution functions associated to $Law_X$, then the combinator $choose(X, Law_X, Process)$ is stochastic bound consistent iff*

$$\exists f_1, f_2 \in \mathcal{F}_X \text{ such that } f_1(u) = min(X) \text{ and } f_2(u) = max(X)$$

From the definitions, it is clear that *stochastic domain consistency* implies *stochastic bound consistency.*

### 3.3 Filtering on $dom(X)$

Given an uniform random value, the principle of the filtering algorithm is to detect values for the stochastic variable than cannot be randomly chosen.

**The algorithm.**
For a given $u$, reasoning on the set of possible probability distributions $\mathcal{F}_X$ and the set of probability distributions $\mathcal{SL}_X$ yields to prune $dom(X)$. For each value $v$ of $dom(X)$, we compute a support for the event $X = v$, i.e the set of values for $u$ such as $v$ can be randomly chosen. Formally, this support is defined as follows.

**Definition 5.** *(Support of $X = v$)*
*Let $i$ be the indice of $v$ in $[v_1, \ldots, v_n]$*

$$\forall i \in [1, \ldots, n],\ Supp_{\{X=v\}} = \bigcup_{[v_1 \ldots, v_n] - [w_1, \ldots, w_n] \in \mathcal{SL}_X} \left[ \frac{\sum_{j=1}^{i-1} w_j}{\sum_{j=1}^{n} w_j},\ \frac{\sum_{j=1}^{i} w_j}{\sum_{j=1}^{n} w_j} \right[$$

Our filtering algorithm is based on the following property:

$$\forall v \in [v_1, \ldots, v_n],\ u \notin Supp_{\{X=v\}} \Rightarrow X \neq v$$

However, the computation of $Supp_{\{X=v\}}$ is based on the labelling of each element of $\mathcal{SL}_X$ and in the worst case, the number of elements of $\mathcal{SL}_X$ is exponential with the size of the probabilistic choice. Indeed, this is the cardinality of the Cartesian product of the domains of the weight variables. As a consequence, we propose to approximate the computation of $Supp_{\{X=v\}}$ using only the bounds of this union of intervals. This is shown on Algorithm 1 that presents our filtering algorithm. It takes as inputs $X$, $Law_X$ and $u$ and removes values from $dom(X)$ that cannot be randomly chosen. For each value $v \in dom(X)$, the bounds of $Supp_{\{X=v\}}$, noted $Min_Sup_{\{X=v\}}$ and $Max_Sup_{\{X=v\}}$, are computed by the function IntProba. If $u \notin [Min_Supp_{\{X=v\}}; Max_Supp_{\{X=v\}}]$, then $v$ is removed from $dom(X)$.

The efficiency of the filtering algorithm relies directly on the efficient computation of the bounds of each support $Supp_{\{X=v\}}$. Fortunately, theses bounds can be computed from the result of an analytical study that is described below.

**Algorithm 1:** FilteralGo

```
Input  : X, Law_X and u
forall v ∈ dom(X) do
    [Min_Sup_{X=v}; Max_Sup_{X=v}] ← IntProba(v, Law_X);
    if u ∉ [Min_Sup_{X=v}; Max_Sup_{X=v}] then
        v is removed from dom(X)
    end
end
```

**A method to compute an approximation of $Supp_{\{X=v\}}$.**
The method aims at computing the two elements of $\mathcal{SL}_X$ such that $Supp_{\{X=v\}}$ is minimized and maximized. For that, we compute the minimum (resp. maximum) value of $\frac{\sum_{j=1}^{i-1} w_j}{\sum_{j=1}^{n} w_j}$ (resp. $\frac{\sum_{j=1}^{i} w_j}{\sum_{j=1}^{n} w_j}$) for each distribution probability of $\mathcal{S}L_X$ where $i$ denotes the indice of $v$ in $[v_1, \ldots, v_n]$. Computing $\min_{[v_1 \ldots, v_n]-[w_1, \ldots, w_n] \in \mathcal{SL}_X} \left( \frac{\sum_{j=1}^{i-1} w_j}{\sum_{j=1}^{n} w_j} \right)$ and $\max_{[v_1 \ldots, v_n]-[w_1, \ldots, w_n] \in \mathcal{SL}_X} \left( \frac{\sum_{j=1}^{i} w_j}{\sum_{j=1}^{n} w_j} \right)$ is efficient by using the two following analytical results.

Without any loss of generality, we suppose that $\sum_{j=1}^{n} min(W_j) > 0$. Then,

$\forall i \in [1, \ldots, n]$,

$$\min_{[v_1 \ldots, v_n]-[w_1, \ldots, w_n] \in \mathcal{SL}_X} \left( \frac{\sum_{j=1}^{i-1} w_j}{\sum_{j=1}^{n} w_j} \right) = \frac{\sum_{j=1}^{i-1} \min(W_j)}{\sum_{j=1}^{i-1} \min(W_j) + \sum_{j=i}^{n} \max(W_j)}$$

and

$\forall i \in [1, \ldots, n]$,

$$\max_{[v_1 \ldots, v_n]-[w_1, \ldots, w_n] \in \mathcal{SL}_X} \left( \frac{\sum_{j=1}^{i-1} w_j}{\sum_{j=1}^{n} w_j} \right) = \frac{\sum_{j=1}^{i} \max(W_j)}{\sum_{j=1}^{i} \max(W_j) + \sum_{j=i+1}^{n} \min(W_j)}$$

The complete proofs of these equations can be done by refutation and is available in [11].

Consider again the example 2 and the set $\mathcal{F}_X$ given in FIG. 3. The values of $Supp_{\{X=v\}}$ are computed by our algorithm and are shown below.

| | |
|---|---|
| $Supp_{\{X=1\}} = [0 \ ; \frac{1}{6}[$ | $Supp_{\{X=4\}} = [\frac{5}{13} \ ; \frac{2}{3}[$ |
| $Supp_{\{X=2\}} = [\frac{1}{13} \ ; \frac{1}{3}[$ | $Supp_{\{X=5\}} = [\frac{7}{13} \ ; \frac{5}{6}[$ |
| $Supp_{\{X=3\}} = [\frac{3}{13} \ ; \frac{1}{2}[$ | $Supp_{\{X=6\}} = [\frac{9}{13} \ ; 1[$ |

$X = 1$ is removed from $dom(X)$ by the filtering algorithm when $u \notin [0; \frac{1}{6}[$, $X = 2$ is removed from $dom(X)$ when $u \notin [\frac{1}{13} \ ; \frac{1}{3}[$ and so on.

**Termination and correction**
The filtering algorithm iterates over the possible values of $dom(X)$ then its termination is trivially demonstrated, as the set $dom(X)$ is finite.

Correction of the algorithm can be proved by showing that it permits to achieve a chosen level of consistency. Based on our definition 3 and 4 of stochastic consistencies , we show that FILTERALGO achieves only *stochastic bound consistency*. Due to the approximation of $Supp_{\{X=v\}}$, *stochastic domain consistency* cannot be achieved by our algorithm.

We only give a sketch of the proof that is available in [11]. We have to prove that after the execution of FILTERALGO, there exist two distribution functions $f_1$ and $f_2$ of $\mathcal{F}_X$ such that $f_1(u) = min(X)$ and $f_2(u) = max(X)$. Given a value $v_j$ for the minimum (resp. the maximum) of $dom(X)$ in $[v_1, \ldots, v_n]$, the proof

consists in finding an element of $\mathcal{SL}_X$ $[w_1, \ldots, w_{j-1}, w_j, \ldots, w_n]$ such that $u \in \left[\frac{\sum_{i=1}^{j-1} w_j}{\sum_{i=1}^{n} w_j}, \frac{\sum_{i=1}^{j} w_j}{\sum_{i=1}^{n} w_j}\right[$. A case-based reasoning on the distinct values of $min(X)$ and $max(X)$ in $[v_1, \ldots, v_n]$ completes the proof.

**Complexity**
The complexity of the filtering algorithm relies on the computation of $Supp_{\{X=v\}}$ and each bound of $Supp_{\{X=v\}}$ is obtained by computing $\forall i \in 1 \ldots n$, $\sum_{j=1}^{i} min(W_j)$ and $\sum_{j=1}^{i} max(W_j)$. So, the results are computed in a linear time w.r.t. the size of $dom(X)$.

As the rest of the algorithm is restricted to a "membership testing", FILTERALGO runs in $O(n)$ where $n$ is the size of the domain of $X$.

## 4 Implementation

In this section, we describe our implementation of the filtering algorithm within several PCOs. This implementation is based on the library PCC(FD) [10] that contains three PCOs defined using the global constraint interface of SICStus Prolog [1]. These three combinators are based on $choose(X, Law_X, Process)$, but distinguish themselves by the definition of $Law_X$. In all the cases, the probabilistic choice $Domain - Distribution$ takes the form of Prolog terms compound of unbound parameters that model uncertainty. These combinators are as follows:

- `choose`, where *Domain* is a list of values and *Distribution* is a list of finite domain variables that represent weights;
- `choose_range`, where *Domain* is a range represented with two distinct FD variables $Min$ and $Max$, and *Distribution* is a list of finite domain variables that represent weights;
- `choose_decision`, where *Domain* is the boolean domain $\{0, 1\}$ and *Distribution* is a pair of distinct finite domain variables that represent weights.

When a PCO is posted in the constraint store, the predicate `random/1` allows to obtain a uniform pseudo-random value for $U$. During the constraint propagation, PCOs `choose`, `choose_range` and `choose_decision` are awoken when the domain of the stochastic variable $X$ or the domain of weight variables are pruned. Then, the filtering algorithm is launched as soon as new information on the probabilistic choice is available.

Some options are available to parameterize the filtering capabilities of the algorithm. The `domain_bound` option is used to launch our implementation of FILTERALGO, while the `no_filtering` option permits to switch off the filtering algorithm. The `inconsistency_check` option checks whether *Process* is partially consistent w.r.t. $dom(X)$. This option is useful to improve the pruning capabilities of the filtering algorithm but is also more costly.

## 5 Experimental validation

In this section, we present the experimental results we obtained on two applications of PCOs. The first one concerns biased games models that play a prevalent role in many applications. The second one is a resource planning application adapted from [16].

### 5.1 Biased games model

In this subsection, we consider two dice games: 421 and *less than* 8. We start by briefly describing the models of the games and then we present the experimental results.

**Models.**
**421.** The 421 game consists in drawing three N-face dice. The game is won when faces 4, 2 and 1 are drawn without considering the valuation order of the dice. The problem we adress here is to play 421 with a biased die for which the exact bias is unknown.

A model that considers only a 6-face die is given below by the predicate `four_two_one/3`. In the model, the unknown bias is formalized via the Finite Domains variables $[W1, .., W6]$ representing the weights of the probability distribution. Each weight has a value in 1..10 and additionnal constraints on the bias are given by the constraints `2*W1#=W6, 2*W2#=W5 ,2*W4#=W3`.

```
four_two_one([D1,D2,D3],[W1,W2,W3,W4,W5,W6],Opt) :-
       domaine([W1,W2,W3,W4,W5,W6],1,10),
       2*W1=W6, 2*W2=W5 ,2*W4=W3,
       choose(D1, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
       choose(D2, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
       choose(D3, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
       all_different([D1,D2,D3]),
       D1+D2+D3#=7.
```

The predicate `four_two_one([D1,D2,D3],Distribution,Opt)` is true iff `D1`, `D2` and `D3` have been valuated to 4, 2 and 1, which is modelled by the two latter constraints.

**Less Than 8.** This game consists in drawing five N-face dice. The game is won when the sum of the values of the dice are less or equal than 8. The game with 6-face dice is modelled by `less_than_8/2` predicate.

```
less_than_8([D1,D2,D3,D4,D5],[W1,W2,W3,W4,W5,W6],Opt) :-
       domain[W1,W2,W3,W4,W5,W6],1,10),
       4*W6 #= W1,3*W5 #= W2,2*W4 #= W3,
       choose(D1, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
       choose(D2, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
       choose(D3, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
```

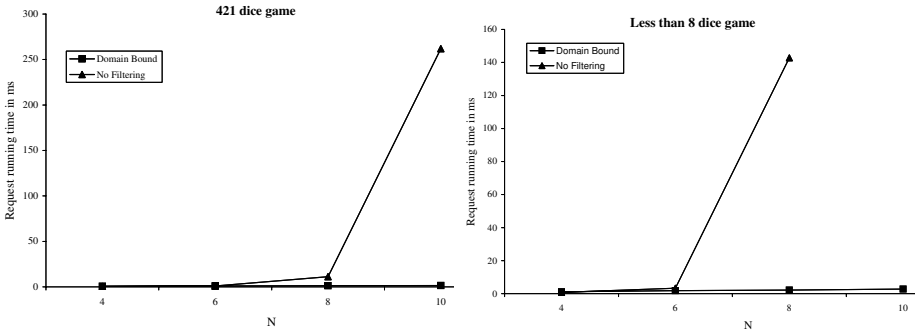


**Fig. 4.** Experimental results on 421 and *less than* 8

```
choose(D4, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
choose(D5, [1,2,3,4,5,6]-[W1,W2,W3,W4,W5,W6],[],Opt),
D1+D2+D3+D4+D5 #=<8.
```

**Results**
Our experiment seeks for a valuation of `Distribution` (the bias of the die) such as the game is always won by the player. We parameterized our model with $N$, the number of the faces of the die in order to evaluate the boosting of our filtering algorithm. For both games, the request was iterated 2000 times in order to avoid the factor of good luck due the choice of the uniform random value. A large number of iterations has been chosen in order to get a significant sampling of request. Experimental results were obtained on a Win XP machine powered by a 2GHz Intel Pentium with 1Go memory running SICStus 3.11. We compare two approaches: in the first one, no filtering was launched during constraint propagation (no_filtering) while in the second one, our filtering algorithm was applied (domain_bound).

The results are given in FIG.4. Both curves with the no_filtering option present a combinatorial explosion when $N$ grows while they remain proportional when option domain_bound is selected. This confirms that our filtering algorithms boosts combinators when additional constraints on the probabilistic choice are available. Note that the overhead due to our algorithm during constraint propagation remains modest. Note also that our results are nearly the same regardless the labelling heuristic that is used. This comes from the symmetry of the problems.

### 5.2 Books production planning

**Models.**
Our books production planning constraint model is adapted from the classical

$M$ quarter production planning problem of [16]. In our model, the demand of books is modelized by the PCO $choose(Demand, [100, 101, 102, 103, 104, 105] - [W_{100}, W_{101}, W_{102}, W_{103}, W_{104}, W_{105}], [], Opt)$ where the weights $W_i$ denote the unknown probability distribution. So, we solve the problem with an unknown probability distribution but we don't solve the original stochastic constraint system aiming at satisfying the clientele eighty percent of the time. Additional constraints on $W_i$ come from the variability on book demands over the year. For example, during winter, the book editor expects that the demand would be greater and then the probability of sold 105 books is 2 times greater than the probability of sold 100 (`2*W100#=W105`).

```
book_production(M,Opt) :-
    domain([W100,W101,W102,W103,W104,W105],1,10),
    2*W100#=W105,2*W101#=W104,2*W102#=W103,
    choose(Demand,[100,101,102,103,104,105]-
                          [W100,W101,W102,W103,W104,W105],[],Opt),
    Prod#>=Demand, Surplus #= Prod-Demand,
    Cost #= Surplus+Cost_Rest,
    M1 is M-1,
    book_production_rec(M1,Prod_Rest,Surplus,Cost_Rest,Opt),
    append([Prod|Prod_Rest],[W100,W101,W102,W103,W104,W105],Lab_List),
    labeling([minimize(Cost)],Lab_List).

book_production_rec(0,[],_Surplus,0,_Opt).

book_production_rec(M,[Prod|Prod_Rest],Surplus,Cost,Opt) :-
    choose(Demand,[100,101,102,103,104,105]-
                          [W100,W101,W102,W103,W104,W105],[],Opt),
    Prod#>=Demand-Surplus,
    Surplus2 #= Prod-Demand+Surplus,
    Cost #= Prod-Demand+Surplus+Cost_Rest,
    M1 is M-1,
    book_production_rec(M1,Production_Rest,Surplus2,Cost_Rest).
```

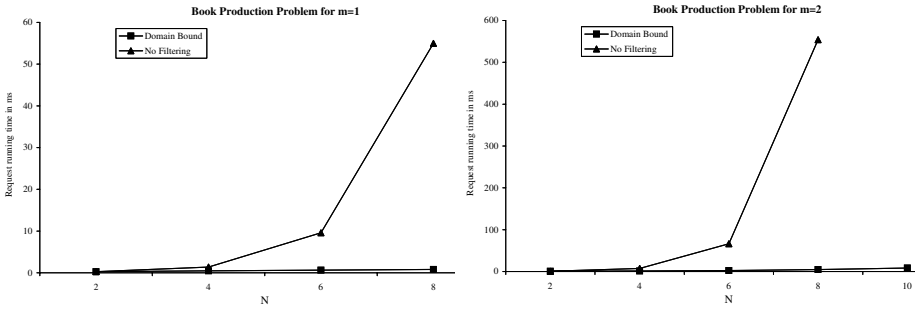


**Fig. 5.** Results on Book Production assignment for $M = 1$ and $M = 2$

**Results.**
In our experiments, two parameters are used to control the model: the variation domain of the book demand $N$ and the number of considered quarters $M$. The experimental results are given by FIG.5. They present a similar profile than the results we got on biased games models. In fact, this is explained by the similar constraints that we use on the weights. However, when changing these constraints in the experiment (removing `2*W1#=W6,2*W2#=W5,2*W3#=W4` and adding `W1+W6#=5`, `W1-W6#=3`,`W2+W5#=5`,`W2-W5=3`,`W3+W4#=5` and `W3-W4#=3`), we got different results that are shown in Fig.6. In this case where the underlying problem is more constrained, the CPU time elapsed in labelling is less important as constraint propagation becomes more efficient. Hence, we suggest to use our filtering algorithm in priority for problems where the probability distribution is weakly constrained and it is difficult to find a bias such as the constraints systems are satisfied.

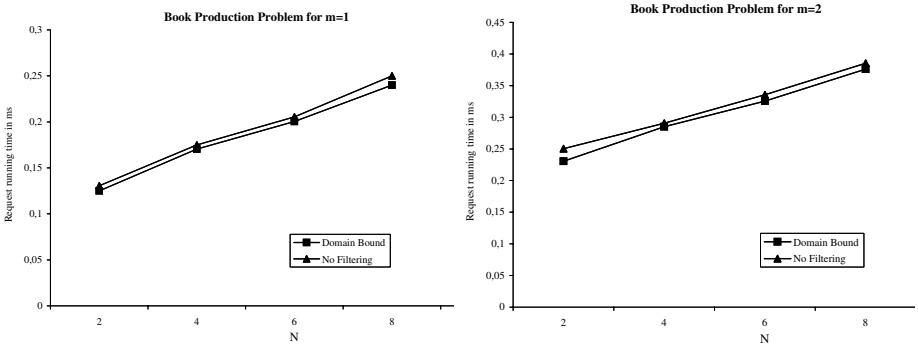


**Fig. 6.** Results on Book Production assignment for a new model of the demand

## 6 Conclusions

In this paper, we have proposed a filtering algorithm associated to combinators that model Probabilistic Choice Operators in the presence of uncertainty. This algorithm exploits the early draw of an auxiliary uniform random value to prune the variation domain of the stochastic variable. Our experimental results obtained on two applications shows that our approach boost the implementation of PCOs as a constraint combinators. Our filtering algorithm runs in $O(n)$ but achieves only *stochastic bound consistency*. It is an open question to know whether efficient (polynomial) algorithms for PCOs can be found to achieve *stochastic domain consistency*. Our future work will be devoted to both extend the application domain of PCOs in the presence of uncertainty and find new algorithms to achieve better level of consistency.

## References


1. M. Carlsson, G. Ottosson, and B. Carlson. An Open–Ended Finite Domain Constraint Solver. In *Proceedings of Programming Languages: Implementations, Logics, and Programs*, 1997.
2. A. Di Pierro and H. Wiklicky. On probabilistic CCP. In *APPIA-GULP-PRODE*, pages 225–234, Grado, Italy, 1997.
3. A. Di Pierro and H. Wiklicky. Implementing randomised algorithms in constraint logic programming. *In Proceedings of the ERCIM/Compulog Workshop on Constraints*, 2000.
4. H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: A probabilistic approach. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 97–104, Grenada, 1993. Springer – Verlag.
5. V. Gupta, R. Jagadeesan, and P. Panangaden. Stochastic processes as concurrent constraint programs. In *Proceedings of Symposium on Principles of Programming Languages*, 1999.
6. V. Gupta, R. Jagadeesan, and V.A. Saraswat. Probabilistic concurrent constraint programming. In *Proceedings of the International Conference Conference on Concurrency Theory*, pages 243–257. Springer, 1997.
7. P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(fd). Technical Report CS-93-02, Brown University, 1993.
8. S. Janson and S. Haridi. Programming paradigms of the Andorra kernel language. In *Proceedings of the International Symposium on Logic Programming*, pages 167–186, San Diego, USA, 1991.
9. M. Petit and A. Gotlieb. Probabilistic choice operators as global constraints : application to statistical software testing. In *Proceedings of International Conference on Logic Programming – Poster Presentation*, LNCS, pages 471–472, 2004.
10. M Petit and A. Gotlieb. Library of probabilistic constraint combinators over finite domain, available at http://www.irisa.fr/lande/petit/tools.html. May 2006.
11. M. Petit and A. Gotlieb. Constraint-based reasoning on probabilistic choice operators. Research Report 6165, INRIA, 04 2007.
12. V.A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of Symposium on Principles of Programming Languages*, pages 333–352, Orlando, Florida, 1991.
13. G. Smolka. The Oz programming model. In *Computer Science Today*, LNCS, vol. 1000, pages 324–343. Springer-Verlag, 1995.
14. S. A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.
15. P. Thévenod-Fosse and H. Waeselynck. An investigation of statistical software testing. *Journal of Sotware Testing, Verification and Reliability*, 1(2):5–25, July 1991.
16. T. Walsh. Stochastic constraint programming. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 111–115, Lyon, France, 2002. IOS Press.
17. N. Yorke-Smith and Gervet C. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, LNCS, pages 769–783, Kinsale, Ireland, 2003. Springer.