Flow-Based Combinatorial Chance Constraints

Andre A. Cire, Elvin Coban, and Willem-Jan van Hoeve

Tepper School of Business, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{acire,ecoban,vanhoeve}@andrew.cmu.edu

Abstract. We study stochastic variants of flow-based global constraints as combinatorial chance constraints. As a specific case study, we focus on the stochastic weighted `alldifferent` constraint. We first show that determining the consistency of this constraint is NP-hard. We then show how the combinatorial structure of the `alldifferent` constraint can be used to define chance-based filtering, and to compute a policy. Our propagation algorithm can be extended immediately to related flow-based constraints such as the weighted cardinality constraint. The main benefits of our approach are that our chance-constrained global constraints can be integrated naturally in classical deterministic CP systems, and are more scalable than existing approaches for stochastic constraint programming.

1 Introduction

Many, if not all, real-world optimization problems contain uncertainty in the input data or in the actual realization of a solution to the problem. Depending on the problem at hand, and on the methodology chosen to solve the optimization problem, this uncertainty can be taken into account in different ways. For example, for call-centers the uncertainty in caller volume and type is critical, and it may be best to apply fixed policies based on analytical methods to route incoming calls. On the other hand, planning the production level at a steel factory involves uncertainty with much lower dynamics, which may be best captured with linear chance constraints. In operations research, the broad area of stochastic programming collects various methodologies for solving optimization problems under uncertainty, including stochastic integer programming [19] and stochastic satisfiability [20].

In constraint programming, uncertainty has received relatively limited attention so far. As argued by Brown and Miguel [5], this may be due to two assumptions that constraint programming makes. First, each problem has a crisp and complete description without uncertainty (i.e., the classical constraint satisfaction problem). Second, problems are not dynamic; they do not change between the initial description and the final execution. Clearly, for many practical problems these assumptions do not hold, but current constraint programming systems offer limited support to address these issues.

Nevertheless, several extensions to constraint programming have been proposed to handle uncertainty, including Probabilistic CSPs [10, 11] and stochastic

constraint programming [29]. More recently, scenario-based stochastic constraint programming [25], and cost-based filtering for stochastic constraint programming [24, 26] have been proposed. Lastly, the concept of *global chance constraints* that was introduced by Rossi et al. [23] is of particular interest to our work.

In this work we extend the work of Rossi et al. [23] by considering global chance constraints that combine random variables (representing the uncertain input data) with a combinatorial structure on the deterministic decision variables. In particular, we consider the chance-constrained version of the `alldifferent` constraint, which is formally defined as the `chance-alldifferent` constraint in Section 3. It is a stochastic variant of the weighted `alldifferent` constraint, where the weight of each variable-value pair is represented by a random variable. The constraint defines that with a given probability, the sum of all weights must be at most a certain threshold value, while at the same time the variables take distinct values.

The chance-constrained `alldifferent` constraint is closely related to *stochastic bipartite matching*, which is broadly applied in real life applications such as economics, healthcare, and wireless communication. Hauskrecht and Upfal [14] analyzed stochastic contract matching problem to find an optimum portfolio out of available buy and sell contracts for commodities. Another application is a stochastic k-set packing problem motivated by a stochastic online matching problem between buyers and commodities in the market [3]. Inspired by the applications in kidney exchange and online dating markets, Chen et al. [7] studied a stochastic matching problem with patience. In the framework of two-stage stochastic optimization with recourse, Katriel et al. [17] analyzed two versions of the bipartite matching problem for commodity trading, reservation systems, and scheduling under uncertainties. Moreover, an online bipartite matching problem was studied in [6] motivated by applications in wireless communication.

An important aspect of our approach is that even though we are formally solving a stochastic constraint programming model, in practice the `chance-alldifferent`, or similar constraints, can be embedded in standard CP systems without fundamental changes to the modeling interface or solving mechanism. Namely, the stochastic information is added as an argument to the constraint and handled internally by the propagator, while the interaction with the rest of the model takes place through the deterministic finite domain variables.

Our contributions are the following. First, we show that deciding the feasibility of the `chance-alldifferent` is NP-hard. Second, we propose dedicated filtering algorithms that enable to remove provable inconsistent values from the domains of the (deterministic) decision variables, as well as from the random variables that represent the problem uncertainty. We show that our algorithms are generic in that they apply immediately to related flow-based constraints such as the weighted cardinality constraint. Lastly, we demonstrate experimentally that our approach improves existing stochastic programming approaches in terms of scalability and memory consumption.

2 Stochastic Constraint Programming and Related Work

A *stochastic constraint program*, or *SCP*, is defined by a tuple $\langle X, S, D, P, C, \theta, L\rangle$ interpreted as follows [29, 23]. The set X contains *decision* variables which define a solution to the SCP. Each variable $x \in X$ is associated to a *domain* $D(x) \in D$, containing the values variable x can take. S is a set of *stochastic* (or *random*) variables, also associated to a domain $D(s) \in D$. However, the stochastic variables are not free to assign and follow a probability distribution $P_s : D(s) \rightarrow [0, 1]$, $P_s \in P$. C is a set of constraints restricting the valid solution tuples. A constraint $h \in C$ that contains at least one stochastic variable is called a *chance constraint*. For each chance constraint h, the parameter $\theta_h \in \theta$ is a threshold value in the interval $[0, 1]$ indicating the minimum satisfaction probability for the chance-constraint h.

Each SCP model is also associated to a set $L = [\langle X_1, S_1\rangle, \ldots, \langle X_m, S_m\rangle]$ of *decision stages*. The sets $X_1, \ldots, X_m \subseteq X$ form a partition of X, and analogously the sets $S_1, \ldots, S_m \subseteq S$ form a partition of S. To solve an *m-stage* SCP, we need to first find an assignment for the variables X_1 such that, given random values for S_1, we can find an assignment for X_2. This assignment of X_2 must be such that, given random values for S_2, we can find an assignment for X_3. This reasoning is then applied repeatedly until an assignment for X_m can be found. This last assignment of X_m must be such that, given random values for S_m, the hard constraints are satisfied and the chance-constraints are satisfied within the given probabilities θ.

The solution of an m-stage SCP is, in general, represented by a *policy tree* [15]. The arcs in such a tree represent values observed for stochastic variables whereas nodes at each level represent the decisions associated with the different stages.

Global chance-constraints are a generalization of global constraints to the context of SCPs [23]. In stochastic programs, it is common to identify simple chance-constraints of the form $\Pr(x \geq R) \geq \theta$, involving a decision variable x and a random variable R. These constraints typically appear as a set. For example, in an inventory model mentioned in [23], one could enforce service level constraints for every period in the planning horizon, or equivalently $\Pr(I_j \geq R) \geq \theta$ for every time j where I_j is on-hand inventory level and R is stochastic demand. It is thus natural to group these constraints in a single *global chance-constraint*, as together they could potentially reveal structures which are suitable for stronger inference methods.

SCPs can be solved in different ways. For example, Walsh [29] presents a complete algorithm based on backtracking and forward checking. This initial work was then extended in [25], allowing for SCPs with multiple chance-constraints. It also provided a reduction of SCP models to deterministic CP problems through a scenario-based view. Essentially, in the approach of [25] all probabilistic scenarios (or a sample set of scenarios) are represented explicitly and linked to the rest of the model using reified constraints.

In Rossi et al. [23] the global chance constraint *serviceLevelRS* was developed for an inventory management problem with a single product and a single stocking location, to reason on inventory levels at each period. A very recent work

of Hnich et al. [15], closely related to our approach, presents a general methodology to compile global chance-constraints using the propagator of their deterministic counterpart as parameter. Moreover, further extensions and comparisons of arc-consistent concepts are also presented. Another related work is [24], which provides cost-based domain filtering for stochastic knapsack constraints, indicating some relationship between chance-constraints and cost/soft-constraints. Lastly, SCPs were also applied to queue design problems, which can be suitable to global chance-constraints [27, 28].

The chance-constrained `alldifferent` constraint is contextualized in the area of stochastic bipartite matching. Stochastic matching problems are usually regarded in a sequential or online context. In sequential problems, as proposed in [8], edges and nodes occur sequentially with a given probability. The goal is to decide how to select edges so as to maximize the reward expectation at the end of the horizon. Online matchings usually model Internet advertisement problems. Several works, such as [12, 16], provide bounds and approximation factors for different policies.

Another variation of stochastic matching is *two-stage stochastic matching*, in which some nodes have to be matched before their stochastic weights are known (first stage). The remaining variables are then assigned after (second stage). The goal is to minimize the weight sum expectation. This is known to be an NP-Hard problem, as proved in [18]. Several approximation factors to this problem are also proposed [2, 9].

3 The Chance-Alldifferent Constraint

Before introducing the chance-constrained `alldifferent` constraint, we first recall the definition of the deterministic weighted `alldifferent` constraint [22].

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of finite domain deterministic variables. Let w be a matrix of given 'weights' $w_{i,d}$ for $i = 1, 2, \ldots, n$ and $d \in D(x_i)$. Let t be a given threshold value. Then the weighted `alldifferent` constraint can be defined as

$$\texttt{cost-alldifferent}(X, w, t) := \texttt{alldifferent}(X) \wedge \sum_{i=1}^{n} w_{i,x_i} \leq t.$$

That is, it restricts the set of feasible solutions to those variable assignments that consist of distinct values and the total weight of which meets the threshold t.[1] However, as argued before, in most practical cases the input data (in this case, the weight matrix w) is uncertain. Incorporating this uncertainty naturally leads to the following definition of the chance-constrained `alldifferent` constraint.

Let W be a matrix of random variables $W_{i,d}$ for $i = 1, 2, \ldots, n$ and $d \in D(x_i)$, with given independent discrete distributions, representing the uncertain

[1] Observe that for the weighted `alldifferent` constraint the threshold t can be a variable, in general.

weights. Furthermore, let α be a given constant between 0 and 1. We define the `chance-alldifferent` as:

$$\texttt{chance-alldifferent}(X, W, t, \alpha) := \texttt{alldifferent}(X) \wedge \Pr\left(\sum_{i=1}^{n} W_{i,x_i} \leq t\right) \geq \alpha. \quad (1)$$

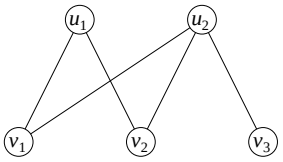
It is well-known that a solution to the `alldifferent` constraint corresponds to a maximum matching in the bipartite 'value graph' $G(X) = (X \cup D, E)$ where $D = \cup_{x \in X} D(x)$ and $E = \{(x, d) \mid x \in X, d \in D(x)\}$ [21]. Similarly, for the `chance-alldifferent` constraint, a solution corresponds to a maximum matching in G such that the total edge weight is at most t, with probability at least α:

Lemma 1. *Let C be* `chance-alldifferent`(X, W, t, α)*. A variable assignment $(x_1, \ldots, x_n) = (d_1, \ldots, d_n)$ is a solution to C if and only if the set $\{(x_i, d_i) \mid i \in \{1, \ldots, n\}\}$ is a matching in $G(X)$ and $\Pr\left(\sum_{i=1}^{n} W_{x_i,d_i} \leq t\right) \geq \alpha$.*

Proof. Immediate from definition (1) and the definition of the value graph. □

Example 1. Consider the following illustrative buy-seller problem, in which a set of traders $S_1 = \{u_1, u_2\}$ wishes to buy contracts from $S_2 = \{v_1, v_2, v_3\}$. We are required to assign one contract to each trader. All possible pairs are allowed, except for pair (u_1, v_3) that has been excluded (see Fig. 1a).

We wish to find matchings with a high total gain, and we will model this by limiting the total loss. The loss for the allowed pairs (u, v), $u \in S_1, v \in S_2$ is given as the discrete probability distribution in Fig. 1b, and collected in the matrix W (the probability of each outcome is indicated in parentheses).



(a) Possible combinations

W_{u_1,v_1} = { 5 (0.2), 9 (0.8) }
W_{u_1,v_2} = { 5 (0.3), 14 (0.7) }
W_{u_2,v_1} = { 6 (0.5), 15 (0.5) }
W_{u_2,v_2} = { 1 (0.9), 18 (0.1) }
W_{u_2,v_3} = { 6 (0.5), 15 (0.5) }

(b) Probability distributions

Fig. 1: The possible combinations and probability distributions W for Example 1.

Let the deterministic decision variable x_i represent the matched element from S_2 for each $i \in S_1$. Thus, $D(x_{u_1}) = \{v_1, v_2\}$ and $D(x_{u_2}) = \{v_1, v_2, v_3\}$. Let $X = \{x_{u_1}, x_{u_2}\}$. We can impose that the total loss must be at most 20, with probability at least 0.8, by posting the constraint

$$\texttt{chance-alldifferent}(X, W, 20, 0.8).$$

Observe that the variable assignment $x_{u_1} = v_1$, $x_{u_2} = v_2$ is a feasible solution. Namely, it respects the `alldifferent` constraint, and moreover

$$\Pr\left(W_{u_1,v_2} + W_{u_2,v_1} \leq 20\right) = 0.2 * 0.9 + 0.8 * 0.9 = 0.9 \geq 0.8,$$

where the terms on the right-hand side correspond, respectively, to the probabilities of the weight pairs $(W_{u_1,v_2}, W_{u_2,v_1}) = (5,1)$, $(9,1)$. On the other hand, the variable assignment $x_{u_1} = v_2$, $x_{u_2} = v_1$ is not feasible, as the valid weight pairs $(W_{u_1,v_2}, W_{u_2,v_1}) = (5,6)$, $(5,15)$, $(14,6)$ yield $\Pr\left(W_{u_1,v_2} + W_{u_2,v_1} \geq 20\right) = 0.65 < 0.8$. □

We note that the definition of `chance-alldifferent` can be readily extended to any weighted global constraint in which the stochastic weights are defined on variable-value pairs.

4 Hardness of Determining Consistency

In this section, we show the following.

Theorem 1. *Deciding whether an arbitrary* `chance-alldifferent` *constraint has a solution is NP-hard.*

Proof. We show that the *K-th Largest m-Tuple* problem ([SP21] in [13]), or `KM`, is a special case of `chance-alldifferent`. The `KM` is defined as follows. Given m sets $X_1, \ldots, X_m \subseteq Z^+$ and positive integers K and B, we want to find K or more distinct m-tuples $(x_1, \ldots, x_m) \in X_1 \times \cdots \times X_m$ for which $\sum_{i=1}^{m} x_i \geq B$.

We now construct an instance of `chance-alldifferent` to solve `KM`. We define variables $\{v_1, \ldots, v_m\}$ with domains $D(v_i) = \{u_i\}$ for $i = 1, \ldots, m$. Notice that there exists only one variable assignment A: $v_i = u_i$ for $i = 1, \ldots, m$. For each pair (v_i, u_i), $1 \leq i \leq m$, define a stochastic domain $D_i = X_i$ where each element in X_i has probability $p_i = 1/|X_i|$. All possible realization scenarios of the assignment A have the same probability, which is given by $\alpha = \prod_{1 \leq i \leq m} p_i$.

Finally, we formulate an instance of `chance-alldifferent` with the variables and domains above, and the constraint

$$\texttt{chance-alldifferent}(v_1, \ldots, v_m, p, B, \alpha K). \tag{2}$$

Since every scenario has probability α, this instance is satisfiable only if there exists at least K scenarios such that the sum of the weights are greater than or equal to B. But each scenario corresponds to an m-tuple of $X_1 \times \cdots \times X_m$ by construction. The theorem then follows. □

We note that although the two-stage stochastic matching problem is known to be NP-Hard, as shown in [18], we were not able to directly use that fact to show the hardness of our particular structure. Also, we are not aware if the problem of deciding whether there exists a feasible solution to `chance-alldifferent` is in NP.

Theorem 1 indicates that it is worthwhile to invest in incomplete filtering methods for the `chance-alldifferent` constraint, that do not necessarily achieve domain consistency. We developed two distinct propagation algorithms that, given a partial variable assignment, help eliminating infeasible stochastic domain values as well as inconsistent values from the domains of deterministic decision variables. These algorithms are described in the next section.

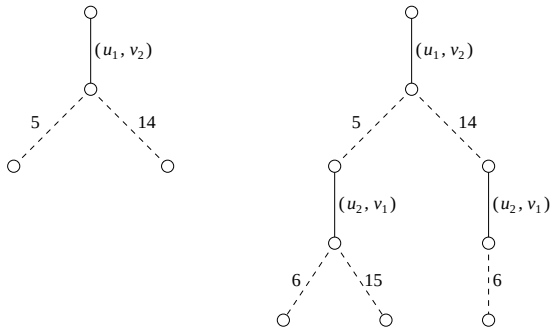
5 Filtering the Chance-Alldifferent Constraint

5.1 Policy Tree Representation

The key idea in our methodology is to cast `chance-alldifferent` as an n-stage stochastic problem. However, we take advantage of the fact that the `chance-alldifferent` constraint does not contain temporal relations, contrary to existing approaches such as n-stage problems in inventory management.

A solution to an n-stage problem is usually defined by means of a *policy tree*, as described in Section 2. In our case, the policy tree will represent all decision variables that have been fixed to a singleton, and the (allowed) realizations of their corresponding stochastic weights. That is, it is a layered graph with at most $2n+1$ layers: Each layer corresponds to a deterministic variable assignment, and the possible weight realizations. Each node (state) in the tree will be assigned the total accumulated weight so far, and the accumulated probability of reaching that state. The root node of the policy tree is a state with total value 0 and probability 1. Let u be a node at level i, representing assignment $x_i = j$ for some $j \in D(x_i)$, with value v_u and probability p_u. We create $|D(W_{i,j})|$ child nodes, where for each $e \in D(W_{i,j})$, the associated node has value $v_u + e$ and probability $p_u \cdot \Pr(e)$, where $\Pr(e)$ represents the probability of e. We can remove from the policy tree all nodes that do not lead to a total value of at most t. The policy tree thus certifies the feasibility of a solution, as the total probability (the sum of the leaf nodes) of the full variable assignment must be at least α. We next illustrate these concepts on Example 1.

Example 2. (Continuing Example 1.) As we have two variables in our example, we have two stages. If we fix the assignments (u_1, v_2) and (u_2, v_1) in this order, the possible stages we analyze during search are presented in Figure 2. Each stage is composed by the fixed assignments so far and their valid realizations. In stage 1 (Figure 2a), we have selected (u_1, v_2) (solid edge) and its possible weight realizations are 5 and 14 (dashed arcs). In stage 2, we extend the tree with (u_2, v_1), as shown in Figure 2b. Since this completes the variable assignment, we need to check if it defines a feasible solution. For this purpose, we compute the weights and probabilities of each leaf of the tree, since the leaf indicates a complete realization of the random variables. The weights are computed by summing up the values of the dashed arcs, while the probabilities are the product of the probability of these values. We can eliminate all leaves with weight more than t, and then verify if the sum of the leaf probabilities is above α. In this example, we removed the realization of 15 for the right-hand edge (u_2, v_1). The



(a) Stage 1 (b) Stage 2

Fig. 2: Two stages for Example 2, when fixing edge (u_1, v_2) (stage 1) and edge (u_2, v_1) (stage 2). Solid arcs correspond to decision variables, while dashed arcs correspond to stochastic variable realizations. The total weight must be at most 20.

sum of the remaining leaf probabilities is 0.65, from which we conclude that the assignment is infeasible. □

In principle, our policy tree can be defined for any order of the variables. However, we propose to follow the order in which variables have been assigned a fixed value. This has the advantage that for search strategies that are based on variable assignments, the policy tree can be updated incrementally (only the last layers have to be removed upon backtracking to an earlier state in the search tree). Alternatively, it suffices to represent only the leaves of the policy tree, as it has the Markov property, *i.e.,* we can generate new leaves by only considering information from the leaves at the current stage of the algorithm. This saves memory, but requires to apply a recomputation upon backtracking.

5.2 Filtering Based on Minimum-Cost Network Flows

Let us first recall some basic definitions from network flow theory [1]. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of finite domain variables. The 'value network' of X is defined as a directed graph $G = (V, A)$ with node set $V = X \cup D \cup \{s, t\}$ where s represents the source and t the sink. The arc set is defined as $A = \{(s, x) \mid x \in X\} \cup \{(x, d) \mid x \in X, d \in D(x)\} \cup \{(d, t) | d \in D\}$. Arcs $a \in \{(s, x) \mid x \in X\}$ have lower capacity $l(a) = 1$ and upper capacity $u(a) = 1$, while arcs $a \in \{(x, d) \mid x \in X, d \in D(x)\} \cup \{(d, t) \in d \in D\}$ have lower capacity $l(a) = 0$ and upper capacity $u(a) = 1$. A flow is a function $f(A) \to \mathbb{R}_{\geq 0}$ such that $\sum_{(i,j)\in A} f(i, j) = \sum_{(j,k)\in A} f(j, k)$, for all $j \in V \setminus \{s, t\}$. A flow f is feasible if $l(a) \leq f(a) \leq u(a)$ for all $a \in A$. A weighted value network $G = (V, A, w)$ has an arc weight function $w : A \to \mathbb{R}$. The weight of a flow f is $w(f) = \sum_{a\in A} w(a)f(a)$. A minimum-cost flow in G is a feasible flow with minimum weight. If all capacities are integer and a feasible flow exists, then also an integer feasible flow exists [1]. We therefore assume in the remainder that flows are binary in our case.

Given a flow f, the *residual* value network $G_f = (V, A_f)$ is defined on the arc set $A_f = \{(i,j) \mid (i,j) \in A, f(i,j) = 0\} \cup \{(j,i) \mid (i,j) \in A, f(i,j) = 1\}$. Furthermore, for a weighted value network $G = (V, A, w)$, the residual weights are defined as $w_f(a) = w(a)$ if $f(a) = 0$, and $w_f(a) = -w(a)$ if $f(a) = 1$.

The next lemma provides a filtering rule based on a minimum weight flow in the value network. For this, we first define the edge weight function $w_{\min}(a) = \min\{D(W_{j,d})\}$ if $a = (x_j, d) \in \{(x,d) \mid x \in X, d \in D(x)\}$, and 0 otherwise.

Lemma 2. *The constraint* `chance-alldifferent`(X, W, t, α) *is inconsistent if no feasible flow exists in* $G = (V, A, w_{\min})$*, or if the total weight of the minimum-cost flow in* G *exceeds* t*.*

Proof. Immediate from the definition of $w_{\min}$. □

Since Lemma 2 applies a deterministic weighted value graph, we can define a sound filtering algorithm based on shortest path computations in the residual value graph, similar to the weighted cardinality constraint [22].

For a flow f in $G = (V, A, w)$ and $i, j \in V$, we let $\mathrm{SP}_f(i,j)$ represent the weight of the shortest i-j path in G_f, where the weight of such path P is defined as $\sum_{a \in P} w_f(a)$ if it exists, and ∞ otherwise.

Lemma 3. *For* `chance-alldifferent`(X, W, t, α)*, let* f *represent a minimum-cost flow in* $G = (V, A, w_{\min})$*, if it exists. For all arcs* $(x_i, d) \in \{(x,d) \mid x \in X, d \in D(x)\}$ *and all* $e \in D(W_{i,d})$*, if*

$$e > t - w(f) - \mathrm{SP}_f(d, x_i),$$

then e *is inconsistent with respect to* `chance-alldifferent`(X, W, t, α)*.*

Proof. The expression $e > t - w(f) - \mathrm{SP}_f(d, x_i)$ stems from evaluating the marginal weight increase when arc (x_i, d) is used with realization $w(x_i, d) = e$. The weight of the minimum-cost flow subject to $f(x_i, d) = 1$ and $w(x_i, d) = e$ is equal to the $w(f) + \mathrm{SP}_f(j, i) + e$. Hence, if this value exceeds t, or $e > t - w(f) - \mathrm{SP}(d, x_i)$, e is inconsistent, by Lemma 2. □

Observe that Lemma 3 allows two types of filtering. First, inconsistent realizations from the stochastic domains can be removed. Second, if a stochastic domain $D(W_{i,d})$ becomes empty, we can remove d from $D(x_i)$. We propose to apply Lemma 3 with respect to the leaves of the policy tree, each of which represents a partial variable assignment with fixed realization of the corresponding stochastic weights. For each leaf node we compute a minimum-cost flow restricted to the value network associated with that node, and perform the domain filtering. Note that because the leaf represents fixed variables and fixed realizations, we can effectively discard those from the network, and compute the flow only with respect to the remaining free variables.

Each application of Lemma 3 first requires the computation of one minimum-cost network flow, which takes $O(n(m + n \log n))$ time when applying the successive shortest path algorithm [1]. Then, for each edge (x_i, d), we only need one

iteration to update $\max\{W_{i,d}\} \leq t - w(f) - \mathrm{SP}(d, x_i)$. For this, we can compute the shortest paths between all nodes in D and X in total $O(|D|\,(m + n \log n))$ time. We remark that the residual networks can be maintained incrementally between propagation events. Moreover, these time complexities are independent of the number of stochastic domain elements.

5.3 Filtering Based on Most Likely Solutions

We next describe a filtering rule based on the following idea. We compute, for each leaf node in the policy tree, an upper bound on the probability of finding solutions completing that node, of value at most t. If the sum of the upper bounds of all leaves is less than α, then the `chance-alldifferent` cannot be satisfied.

In order to find the most likely solution to an instance of `chance-alldifferent`, we extend the value network of Section 5.2 with arc weights $w_{\mathrm{most}}(a) = -\log(\max\{\Pr(e) \mid e \in D(W_{i,d})\})$ if $a = (x_i, d) \in \{(x, d) \mid x \in X, d \in D(x)\}$, and 0 otherwise. In parallel, we maintain the corresponding weights $w'(a) = \arg\max\{\Pr(e) \mid e \in D(W_{i,d})\}$ if $a = (x_i, d) \in \{(x, d) \mid x \in X, d \in D(x)\}$, and 0 otherwise. In case of ties, we let $w'(a)$ be the largest value.

Lemma 4. *A minimum-cost flow f in $G = (V, A, w_{\mathrm{most}})$ corresponds to a variable assignment with maximum total probability that satisfies* `alldifferent`(X).

Proof. Similar to Lemma 2, we know that a feasible flow in G corresponds to a solution to `alldifferent`(X). The maximum total probability of a variable assignment for X is given by the function $T(X) = \prod_{i=1}^{n} \max\{\Pr(e) \mid e \in W_{i,x_i}\}$. As $T(X)$ is increasing, we can instead maximize $\log T(X)$, which is equivalent to maximizing $\sum_{i=1}^{n} \log(\max\{\Pr(e) \mid e \in W_{i,x_i}\})$. This is in turn equivalent to minimizing $-\log T(X)$, or minimizing $\sum_{a \in A} f(a) w_{\mathrm{most}}(a)$, for feasible flows f in G. □

For a constraint `chance-alldifferent`(X, W, t, α), let L be the set of leaves in the policy tree. For each leaf $l \in L$, let X_l be the set of fixed variables. We define the restricted value network G_l as $G \setminus X_l$, i.e., removing all nodes in X_l, the nodes corresponding to their assigned values, and the corresponding arcs. We let P_l denote the probability of reaching l, and w_l the total accumulated weight of l.

We next let f_l represent the minimum-cost flow in G_l, with total associated probability $P'_l = \exp(-\sum_{a \in A} f_l(a) w_{\mathrm{most}}(a))$. The total associated weight is denoted by $w'_l = \sum_{a \in A} f_l(a) w'(a)$. Lastly, we define an upper bound U_l on the probability that l can be extended to a solution that satisfies the threshold t, as

$$U_l = \begin{cases} P_l(1 - P'_l) \text{ if } w_l + w'_l > t \\ P'_l \qquad\qquad \text{otherwise.} \end{cases}$$

Lemma 5. *Let C be a constraint* `chance-alldifferent`(X, W, t, α) *and L the set of leaves of its policy tree. C is inconsistent if $\sum_{l \in L} U_l < \alpha$.*

Proof. Consider a leaf node $l \in L$. If $w_l + w'_l > t$, then with probability P'_l, l will not lead to any solution with total weight at most t, by Lemma 4. Thus, we will have valid solutions with at most probability $(1 - P'_l)$. Therefore, $P_l(1 - P'_l)$ is a valid upper bound for the probability that l leads to success. If otherwise $w_l + w'_l \leq t$, we cannot draw such conclusion, and take P_l as a valid upper bound. Lastly, the leaves L represent all possible scenarios for `chance-alldifferent`(X, W, t, α), and therefore if $\sum_{l, \in L} U_l < \alpha$, the constraint cannot be satisfied. □

We can apply Lemma 5 to identify individual inconsistent variable-value combinations. For this, given a minimum-cost flow f_l in G_l, for all arcs $(x_i, d) \in \{(x, d) \mid x \in X, d \in D(x)\}$ and $e \in W_{i,d}$, we extend the definition of U_l to

$$U_l^e = \begin{cases} P_l(1 - e^{-(\sum_{a \in A} f_l(a) w_{\text{most}}(a) + SP(d, x_i) + \log(\Pr(e)))}) \text{ if } w_l + SP'_l + e > t \\ P'_l \qquad \text{otherwise,} \end{cases}$$

where $SP(d, x_i)$ again represents the shortest path in the residual graph, with respect to f_l and w_{most}, while SP'_l represents the associated weight of that same path, with respect to w'.

Lemma 6. *For* `chance-alldifferent`(X, W, t, α)*, let L be the set of leaves in the policy tree. For all arcs $(x_i, d) \in \{(x, d) \mid x \in X, d \in D(x)\}$ and all $e \in D(W_{i,d})$, if*

$$\sum_{l \in L} U_l^e < \alpha$$

then e is inconsistent with respect to `chance-alldifferent`(X, W, t, α)*.*

Proof. Similar to the proof of Lemma 3, U_l^e represents a network flow in which the realization of e is forced in the solution. By Lemma 5, this is a valid upper bound for l, and therefore $\sum_{l \in L} U_l^e$ represents a valid upper bound for all scenarios under which outcome e is realized. □

We note that similar to the application of Lemma 3, it suffices here to compute only one shortest path for each value-variable pair (d, x_i) to remove infeasible elements from $D(W_{i,d})$.

5.4 Extension to Other Flow-Based Constraints

The only assumption we have made in our algorithms is that the constraint is representable as a minimum-cost network flow and variable assignments (x_i, d) appear as arcs in this network. Therefore, the algorithms immediately apply to other chance-constrained versions of weighted global constraints that can be represented by a minimum-cost network flow, including the weighted cardinality constraint [22] and the weighted *same* constraint [4].

6 Computational Results

In this section we compare our proposed method with one of the current technologies considered in this area, namely the scenario-based view [25] discussed in Section 2: Suppose the random variables W representing the weights are associated with K *scenarios*, where each scenario is a realization of all the variables W. Let $W^k_{i,d}$ represent the observed value of $W_{i,d}$ at a scenario k and p_k be the probability of scenario k where $k = 1, \ldots, K$. The `chance-alldifferent` can be written as the following deterministic CSP:

$$\begin{aligned} &\texttt{alldifferent}(X), \\ &z_k = 1 \iff \textstyle\sum_{i=1}^{n} W^k_{i,x_i} \leq t, \;\; k = 1, \ldots, K \\ &\textstyle\sum_{k=1}^{K} p_k z_k \geq \alpha, \\ &z_k \in \{0, 1\}, \;\; k = 1, \ldots, K. \end{aligned} \tag{3}$$

The scenario-based formulation (3) allows us to take full advantage of state-of-the-art constraint solvers. Nevertheless, its memory requirement is impractical for most realistic instances, unless scenario reduction techniques are applied, at the cost of losing completeness. The work of Hnich et al. [15] tackles this requirement issue by reformulating the problem in the space of policy trees, in which variables $\mathcal{PT}$ represent the value of the decision variable at the tree nodes. It also strengthens the propagation by replacing the reified constraints with deterministic versions of the global chance-constraints. However, it still requires the policy tree to be explicitly represented during all stages of the algorithm in terms of the policy tree variables $\mathcal{PT}$.

Our approach differs from the methods above in that it constructs the policy tree during *search*, since we only require the subtree that corresponds to valid realizations (with respect to the threshold) to certify the feasibility of solutions. The advantage is that, by exploring the combinatorial structure of the flow-based constraints, we hopefully generate sufficiently small subtrees that may still be manageable by existing solvers. Nonetheless, this requires us to take into account incomplete scenario information during search, in comparison to the formulation (3) and the approach in [15]. As a result, we expect our filters to be less effective than these methods, but relatively more scalable, in particular for instances where the combinatorial structure of the `chance-alldifferent` plays a key role for the instance feasibility. (In particular, note that for 1-point distributions our approach reduces to an arc-consistency algorithm for weighted `alldifferent`, which is stronger than Formulation (3) for a single scenario).

The behavior outlined above is indicated by the following experiment. We have generated random instances with $|X| = 4$. Variables were first initialized with domain $D(x_i) = \{1, \ldots, 4\}$, and values were removed uniformly at random so that the number of edges in the corresponding value graph was between 16 and 18. Each $W_{i,d}$ was then assigned a two-point distribution. Three types of

distribution were considered in this work. *Case I*: The higher possible value of the weight has high probability (i.e. larger than 0.50); *Case II*: The higher possible value of the weight has low probability (i.e. smaller than 0.50); and *Case III*: The probabilities of high and low possible values of the weights are created randomly. We uniformly at random selected values for t and α, from the minimum $W_{i,d}$ to the sum of all variables in W.

We have experimented our technique and formulation (3) with 75 instances, equally divided among the three types. For formulation (3), we preprocessed instances by eliminating the scenarios for which the sum of the observations were less than the threshold weight. This yielded models with an average size of 64,963 variables and 194,876 constraints. We note that we were not able to model problems with formulation (3) for which the value graph had more than 18 edges, since this would require on average more than 1 million `element` constraints.

Our method and formulation (3) were implemented in C++ using the Ilog CP Optimizer 2 framework, which provided the search control and the all-different propagation. In particular, we fixed a lexicographic search only on variables X for both techniques. Minimum-cost flow were computed using the *Lemon* COIN-OR library. The experiments ran in an Intel Core 2 computer with 3.0 GHz and 8.0GB RAM.

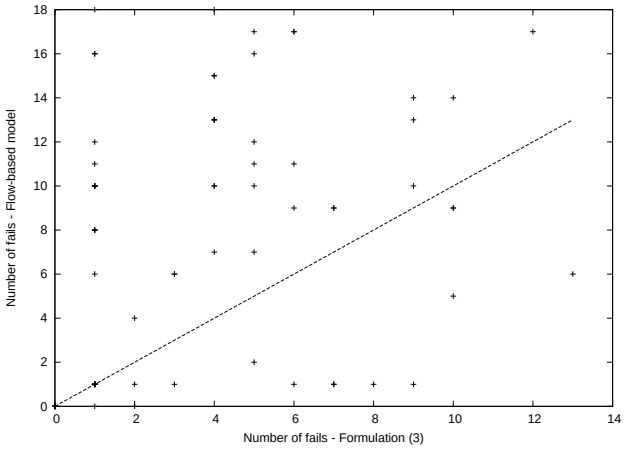


Fig. 3: Number of fails: flow-based approach and formulation (3)

Figures 3 and 4 present scatter plots of the number of fails (i.e., backtracks) and time, respectively, to either find a feasible solution or to prove that the constraint cannot be satisfied. As described earlier, Figure 3 indicates that the filtering provided by the flow-based approach is potentially weaker as the explicit scenario representation of formulation (3). The scenario-based view was particularly effective to perform filtering for Cases 1 and 2, while the cases where flow-based model explored less nodes were concentrated on the random

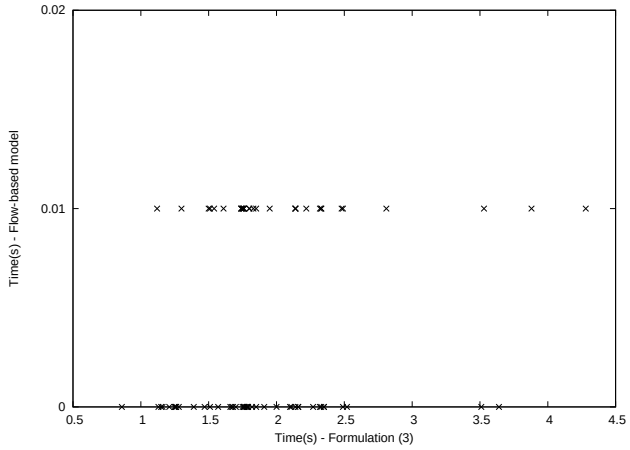


Fig. 4: Time comparison between flow-based approach and formulation (3)

instances. On the other hand, Figure 4 show that all instances were solved in less than 0.01 seconds for the flow-based model using as much as 50MB, while it took on average 1.835 seconds for the scenario-based approach due to the large CSP size.

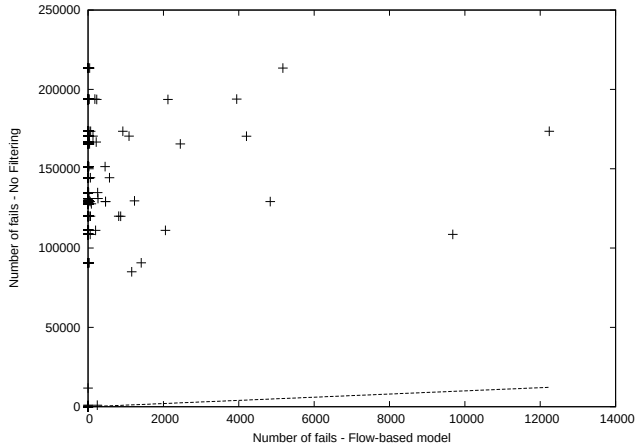


Fig. 5: Number of fails: flow-based approach and formulation for $|X| = 10$

To measure if our approach is scalable to larger domains, we have generated additional 1,344 instances for each case, considering now $|X| = 10$ and value graphs containing between 70 and 80 edges. In this particular experiment, we have only considered a weaker version of Lemmas 3 and 5 for filtering due

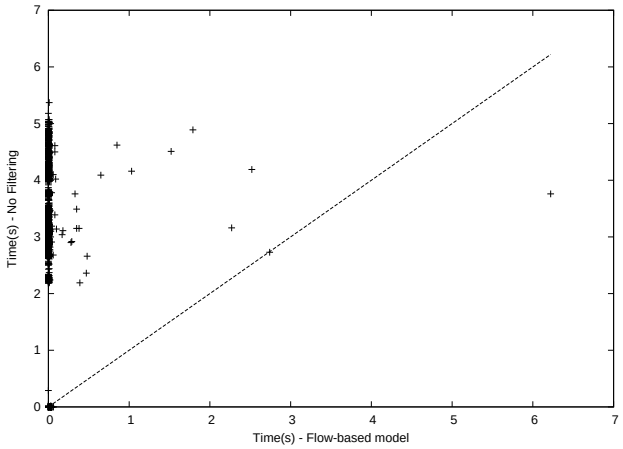


Fig. 6: Time comparison between flow-based approach and formulation for $|X| = 10$

to limits in our implementation. We compared the flow-based approach with a method that fixes a matching and computes the probability that α is satisfied. This method is equivalent to not performing any filtering except for the `alldifferent`. Figures 5 and 6 present scatter plots of the corresponding number of fails and time for such instances, respectively. Figure 5 indicates that the filtering provided by the flow-based approach is stronger than the method without any filtering. As a result, Figure 6 shows that, except for a few instances, the flow-based approach solves the instances in much less time than the method without any filtering.

Finally, we also observe that the flow-based approach was able to solve a few structured instances (Case 1) with $|X| = 25$ and more than 100 vertices. In particular, all instances tested up to this size never exceeded a memory limit of 5 GB. We note that, for such large instances, we might take advantage of the partial information from the policy tree to provide better search strategies, since here only lexicographic ordering was considered.

7 Conclusion and Future Work

We have proposed filtering algorithms for chance-constrained versions of flow-based global constraints, in which the weights are given a discrete stochastic domain. As a particular case study, we focused on the weighted `alldifferent` constraint. We first showed that it is NP-hard to prove consistency for this constraint. However, we proposed partial filtering algorithms based on specific bounding mechanisms that can be computed by means of minimum-cost network flows. We have shown experimentally that our method improves upon existing methods from stochastic constraint programming in terms of memory consumption and scalability.

Bibliography

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] F. Altarelli, A. Braunstein, A. Ramezanpour, and Zecchina R. Stochastic Matching Problem. *Physical Review Letters*, 106(190601), 2011.

[3] N. Bansal, A. Gupta, J. Li, J. Mestre, V. Nagarajan, and A. Rudra. When lp is the cure for your matching woes: Improved bounds for stochastic matchings. In *Proceedings of the 18th Annual European Symposium on Algorithms*, pages 218–230. Springer, 2010.

[4] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering Algorithms for the Same Constraint. In *Proceedings of CPAIOR*, volume 3011 of *LNCS*, pages 65–79, 2004.

[5] K.N. Brown and I. Miguel. Uncertainty and Change. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 21. Elsevier, 2006.

[6] K. Chaudhuri, C. Daskalakis, R. D. Kleinberg, and H. Lin. Online bipartite perfect matching with augmentations. In *INFOCOM*, pages 1044–1052, 2009.

[7] N. Chen, N. Immorlica, A. Karlin, M. Mahdian, and A. Rudra. Approximating matches made in heaven. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 266–278, 2009.

[8] C. Derman, G.J. Lieberman, and S.M. Ross. A Sequential Stochastic Assignment Problem. *Management Science*, 18(7):349–355, 1972.

[9] B. Escoffier, L. Gourvès, J. Monnot, and O. Spanjaard. Two-stage stochastic matching and spanning tree problems: Polynomial instances and approximation. *European Journal of Operational Research*, 205(1):19–30, 2010.

[10] 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*, volume 747 of *Lecture Notes in Computer Science*, pages 97–104. Springer, 1993.

[11] H. Fargier, J. Lang, R. Martin-Clouaire, and T. Schiex. A constraint satisfaction framework for decision under uncertainty. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 167–174. Morgan Kaufmann, 1995.

[12] J. Feldman, A. Mehta, V.S. Mirrokni, and S. Muthukrishnan. Online Stochastic Matching: Beating 1-1/e. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 117–126. IEEE Computer Society, 2009.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., 1979.

[14] M. Hauskrecht and E. Upfal. A clustering approach to solving large stochastic matching problems. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 219–226, 2001.

[15] B. Hnich, R. Rossi, S.A. Tarim, and S.D. Prestwich. Synthesizing Filtering Algorithms for Global Chance-Constraints. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *Lecture Notes in Computer Science*, pages 439–453. Springer, 2009.

[16] C. Karande, A. Mehta, and P. Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 587–596. ACM, 2011.

[17] I. Katriel, C. Kenyon-Mathieu, and E. Upfal. Commitment under uncertainty: Two-stage stochastic matching problems. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming*, volume 4596 of *LNCS*, pages 171–182. Springer, 2007.

[18] N. Kong and A.J. Schaefer. A factor 1/2 approximation algorithm for two-stage stochastic matching problems. *European Journal of Operational Research*, 172(3): 740–746, 2006.

[19] F.V. Louveaux and R. Schultz. Stochastic Integer Programming. In A. Ruszczynski and A. Shapiro, editors, *Stochastic Programming*, volume 10 of *Handbooks in Operations Research and Management Science*. Elsevier, 2003.

[20] S.M. Majercik. Stochastic Boolean Satisfiability. In A. Biere, M. Heule, M. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, pages 887–925. IOS Press, 2009.

[21] J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1, pages 362–367. AAAI Press, 1994.

[22] J.C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7:387–405, 2002.

[23] R. Rossi, S.A. Tarim, B. Hnich, and S.D. Prestwich. A Global Chance-Constraint for Stochastic Inventory Systems Under Service Level Constraints. *Constraints*, 13(4):490–517, 2008.

[24] R. Rossi, S.A. Tarim, B. Hnich, and S.D. Prestwich. Cost-Based Domain Filtering for Stochastic Constraint Programming. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming*, volume 5202 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2008.

[25] S.A. Tarim, S. Manandhar, and T. Walsh. Stochastic Constraint Programming: A Scenario-Based Approach. *Constraints*, 11(1):53–80, 2006.

[26] S.A. Tarim, B. Hnich, R. Rossi, and S.D. Prestwich. Cost-Based Filtering Techniques for Stochastic Inventory Control Under Service Level Constraints. *Constraints*, 14(2):137–176, 2009.

[27] D. Terekhov and J.C. Beck. A constraint programming approach for solving a queueing control problem. *J. Artif. Int. Res.*, 32:123–167, 2008.

[28] D. Terekhov, J.C. Beck, and K.N. Brown. A Constraint Programming Approach for Solving a Queueing Design and Control Problem. *INFORMS Journal on Computing*, 21(4):549–561, 2009.

[29] T. Walsh. Stochastic Constraint Programming. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence*, pages 111–115. IOS Press, 2002.