An improvement of Random Node Generator for the uniform generation of capacities

Peiqi Sun, Michel Grabisch, Christophe Labreuche

An improvement of Random Node Generator for the uniform generation of capacities

Peiqi SUN[1]*, Michel GRABISCH[2] and Christophe LABREUCHE[3].
[1] Université Paris I - Panthéon-Sorbonne, Paris, France
peiqisun94@gmail.com
[2] Paris School of Economics, Université Paris I - Panthéon-Sorbonne, Paris, France
michel.grabisch@univ-paris1.fr
[3] Thales Research & Technology, Palaiseau, France
christophe.labreuche@thalesgroup.com

December 20, 2023

Abstract

Capacity is an important tool in decision-making under risk and uncertainty and multi-criteria decision-making. When learning a capacity-based model, it is important to be able to generate uniformly a capacity. Due to the monotonicity constraints of a capacity, this task reveals to be very difficult. The classical Random Node Generator (RNG) algorithm is a fast-running speed capacity generator, however with poor performance. In this paper, we firstly present an exact algorithm for generating a n elements' general capacity, usable when $n < 5$. Then, we present an improvement of the classical RNG by studying the distribution of the value of each element of a capacity. Furthermore, we divide it into two cases, the first one is the case without any conditions, and the second one is the case when some elements have been generated. Experimental results show that the performance of this improved algorithm is much better than the classical RNG while keeping a very reasonable computation time.

Keywords: random generation, capacity, linear extension

1 Introduction

Capacities and the Choquet integral are widely used in decision making, especially in decision with multiple criteria, where the capacity models the importance of groups of criteria while the Choquet integral is used as a versatile aggregation operator [7, 9]. It is often useful in practice to be able to randomly generate capacities, in a uniform way (measure of performance of models, evaluation of an elicitation technique, learning/identification phase, etc.). This problem reveals to be surprisingly difficult, because

*Corresponding author

of the monotonicity constraints defining capacities, so that naive approaches yield poor performance and give highly biased distributions.

The theoretical perfect solution to the random generation problem is however known: since the set of capacities is an order polytope, generating capacities in a uniform way amounts to generating all linear extensions of the Boolean lattice $(2^N, \subseteq)$ [23]. This leads to the exact capacity generator (ECG), which we have implemented. However, the number of linear extensions of $(2^N, \subseteq)$ grows tremendously fast with $n := |N|$, and is even not known beyond $n = 8$. Therefore, approximate solutions have to be found. One way is to generate a sufficiently representative subset of linear extensions: this is the approach taken by Karzanov and Khachiyan using Markov Chains [14], Combarro et al [4,5], and also the authors of paper [10]. Another way is to find some simple heuristic for directly generating one by one all the coefficients of a capacity, for example, the random node generator of Havens and Pinar [12]. This generator is very fast but has poor performance, due to the fact that for simplicity the coefficients of a capacity are supposed to follow a uniform distribution on some interval. However, the theoretical distribution of a coefficient is very complex and relies also on linear extensions.

The first aim of this paper is to provide an improvement of the random node generator of Havens and Pinar, called IRNG, by taking advantage of some properties of the exact distribution of the coefficients of a capacity. We show that distributions obtained by our method are much closer to the exact distributions or those obtained by the Markov Chain method, while demanding a small computation time, which is much lower than the time required by the Markov Chain method.

Our second aim is motivated by the fact that, in practice, it is often necessary to generate a set of capacities in a uniform way but *subject to some constraints*, which could come from some preference information given by the decision maker, or when using the approach of Stochastic Multiobjective Acceptability Analysis (SMAA) [1, 18]. A naive solution to this problem is the acceptance and rejection method, which amounts to generating capacities and keeping only those which satisfy the constraints. Indeed, the acceptance rate would be in most cases too small to make the method tractable. Therefore, one has to take into account the constraints directly into the generation method. Incorporating arbitrary linear constraints on capacity coefficients into ECG or IRNG seems however to be infeasible. We have therefore restricted to constraints where two capacity coefficients are compared, or one coefficient is compared to some value, and proposed a modification of ECG and IRNG in order to take into account these types of constraints. As preferential information leads in general to more complex constraints than the two types we restrict to, there is still a acceptance and rejection step in our approach, but with a much better acceptance rate than with the naive method.

The paper is organized as follows: Section 2 gives the necessary background on multicriteria decision making, capacities, and the random generation of capacities. Section 3 presents the exact method based on linear extensions (ECG algorithm) and the improvement of the Random Node Generator (IRNG algorithm) based on the study of the theoretical distribution of the coefficients of a capacity. Section 4 is devoted to the experimental results on the comparison of IRNG with the original Random Node

Generator and the Markov Chain method. In Section 5, we study how to incorporate constraints into our ECG and IRNG algorithms. Experimental results are given to show the advantage on the naive approach. Section 6 concludes the paper.

2 Background

2.1 Multi-Criteria Decision Aiding

Multi-Criteria Decision Aiding (MCDA) consists in modeling the preferences of a Decision Maker (DM) regarding alternatives on the basis of multiple and conflicting criteria. We denote by $N = \{1, \ldots, n\}$ the set of criteria. Each criterion $i \in N$ is associated with an attribute X_i, and the alternatives are thus elements of the Cartesian product $X = X_1 \times \cdots \times X_n$. The preference relation of the DM is denoted by $\succsim$ where $x \succsim y$ (for $x, y \in X$) means that the DM prefers x over y. In general, we look for a *numerical representation* [17] $u : X \to \mathbb{R}$ of the preference relation such that:

$$\forall x, y \in X \quad , \quad x \succsim y \;\Leftrightarrow\; u(x) \geq u(y). \tag{1}$$

Without loss of generality, we consider scale $[0, 1]$, where 0 (resp. 1) means that the criterion is not satisfied at all (resp. perfectly satisfactory). Hence $u : X \to [0, 1]$. Function u is often written in a decomposable form [15], where we first normalize the attributes by introducing a utility function mapping each attribute into the satisfaction scale $[0, 1]$, and then aggregate the normalized scores to produce the overall score. These two steps are in general handled separately, and for the purpose of this paper, we only consider the second step, that is the aggregation step. Hence, in order to avoid cumbersome notation, we assume that the input scores are already normalized, that is $X_1 = \cdots = X_n = [0, 1]$ and $X = [0, 1]^N$. Hence $u(x) = F(x_1, \ldots, x_n)$, where $F : [0, 1]^n \to [0, 1]$ is an *aggregation function*.

The most classical aggregation function is the weighted sum $F(x_1, \ldots, x_n) = \sum_{i \in N} w_i x_i$. This model assumes that all criteria are independent, which is often violated in real applications. We consider thus a more versatile aggregation function called the Choquet integral. A *capacity* on N is a set function $\mu : 2^N \to [0, 1]$ such that $\mu(\emptyset) = 0$, $\mu(N) = 1$ (normalization) and $\mu(A) \leq \mu(B)$ whenever $A \subseteq B$ (monotonicity) [6]. The set of capacities on N is denoted by $\mathcal{C}(N)$.

The Choquet integral is a generalization of the weighted sum, taking into account the weights μ [6]:

$$C_\mu(x) = \sum_{k=1}^{n} \big(x_{\tau(k)} - x_{\tau(k-1)}\big)\ \mu(\{\tau(k), \ldots, \tau(n)\}), \tag{2}$$

where τ is a permutation on N such that $x_{\tau(1)} \leq x_{\tau(2)} \leq \cdots \leq x_{\tau(n)}$, and with the notation $x_{\tau(0)} := 0$. Capacity μ can be nicely interpreted as the overall score of a particular alternative:

$$C_\mu(1_B, 0_{N \setminus B}) = \mu(B), \tag{3}$$

where, for $x, y \in X$, $(x_B, y_{N\setminus B})$ denotes an alternative taking the value x_i when $i \in B$ and y_i otherwise. Alternative $(1_B, 0_{N\setminus B})$ is called *binary alternative* and is denoted by a_B.

Capacity μ is elicited from *preference information* provided by the DM. It typically consists of

- a set P of pairs of alternatives $(a, b) \in X^2$ such that a is strictly preferred to b for the DM, and
- a set I of pairs of alternatives $(a, b) \in X^2$ such that a is indifferent to b for the DM.

We consider then the set of capacities that are compatible with the preference information P, I of the DM:

$$\mathcal{C}(N; P, I) = \{\mu \in \mathcal{C}(N) \ : \ C_\mu(a) \geq C_\mu(b) + \varepsilon \ \forall (a, b) \in P \text{ and } C_\mu(a) = C_\mu(b) \ \forall (a, b) \in I\}$$

where $\varepsilon > 0$ is a fixed threshold. Usually, the capacity which is chosen in this set is a solution of an optimization problem under constraint $\mathcal{C}(N; P, I)$ [8], where the objective function to maximize can be for example the entropy of μ [16].

2.2 Need of random capacity generator

The need to have a (unbiased) generator of capacities arise in many problems. Genetic algorithms are classical techniques to learn a capacity [20]. They start from a uniform population of capacities and require thus a random generator of capacities. Another application is for the experimental evaluation of an elicitation technique [13]. To this end, one needs a random generator of capacities, from which one can generate preference information.

Having a generator of (unbiased) capacities compatible with the preferential information is also quite important in decision. This is particularly the case with the recommendation regarding a discrete set $A \subset X$ of options. Ideally, one would like to make a *robust* recommendation, which occurs when there exists $a \in A$ such that for all $b \in A \setminus \{a\}$ and all $\mu \in \mathcal{C}(N; P, I)$, $C_\mu(a) \geq C_\mu(b)$ [2, 11]. Unfortunately, this condition is very strong and is far from being satisfied in most cases. A weaker version consists in counting the proportion of capacities in $\mathcal{C}(N; P, I)$ for which an alternative dominates another one, as depicted in Stochastic Multiobjective Acceptability Analysis (SMAA) [1, 18]. As this ratio cannot be computed theoretically, having a good numerical approximation of this quantity relies on an unbiased random generator of $\mathcal{C}(N; P, I)$. Note that SMAA traditionnally uses the *Hit and Run* random generator of a polytope, which does not provide any guarantee to be unbiased.

2.3 Random generator of linear extensions

Let P be a finite set, endowed with a partial order $\preccurlyeq$. We say that $(P, \preccurlyeq)$ is a *(finite) poset*. We recall the following notions:

- $x \in P$ is *maximal* if $x \preccurlyeq y$ with $y \in P$ implies $x = y$. We denote by $\mathsf{Max}(P, \preccurlyeq)$ (simply $\mathsf{Max}(P)$) the set of maximal elements of P.
- A *linear extension* of $(P, \preccurlyeq)$ is a total order $\leqslant$ on P which is compatible with the partial order $\preccurlyeq$ in the following sense: $x \preccurlyeq y$ implies $x \leqslant y$.
- The *order polytope* [23] associated to $(P, \preccurlyeq)$, denoted by $\mathcal{O}(P)$, is the set

$$\mathcal{O}(P) = \{f : P \longrightarrow [0,1] \mid f(x) \leqslant f(y) \text{ if } x \preccurlyeq y\}.$$

It is known from Stanley [23] that linear extensions induce a triangulation of $\mathcal{O}(P)$ into simplices of equal volume. Therefore, generating in a random uniform way an element of $\mathcal{O}(P)$ amounts to generating all linear extensions, or to generating them randomly according to a uniform distribution.

We apply this result to capacities. It is easy to see that the set $\mathcal{C}(N)$ of capacities is an order polytope, whose underlying poset is $(2^N \setminus \{\varnothing, N\}, \subseteq)$. Therefore, the problem of randomly generating capacities according to a uniform distribution amounts to generating the linear extensions of the poset $(2^N \setminus \{\varnothing, N\}, \subseteq)$. For example, for a 3 elements' capacity, $(\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\})$ is a linear extension of the poset $(2^{\{1,2,3\}} \setminus \{\varnothing, \{1,2,3\}\}, \subseteq)$. However, the number of linear extensions of $(2^N \setminus \{\varnothing, N\}, \subseteq)$ increases tremendously fast, and is unknown beyond $n = 8$.

3 Capacity Generators: Random Node generator based on Beta distribution

3.1 Exact Capacity Generator

When $n \leq 4$, it is possible to have an exact algorithm generating all linear extensions, and therefore to generate capacities in a uniform way. We propose below such an algorithm (**Exact-capacity-generator** (ECG)), which is recursive and performs a Depth-First-Search (DFS) finding maximal elements of a poset, which will form the tail of the list describing the linear extension. The following dendrogram of Figure 1 (right) illustrates the process of the algorithm for a 3 elements' capacity. The maximal element is $\{1,2,3\}$, which is the root of the dendrogram (Figure 1, right), then we continue to find the set of maximal elements of the poset deprived of node $\{1,2,3\}$, which is $\{\{1,2\}, \{1,3\}, \{2,3\}\}$, that is the second level of dendrogram. Next, we continue to find the set of maximal elements when each node in the second level of the dendrogram is removed. We repeat the above steps until there is only one element left in the poset to obtain the whole dendrogram.

Algorithm 1

Exact-capacity-generator(n, k)

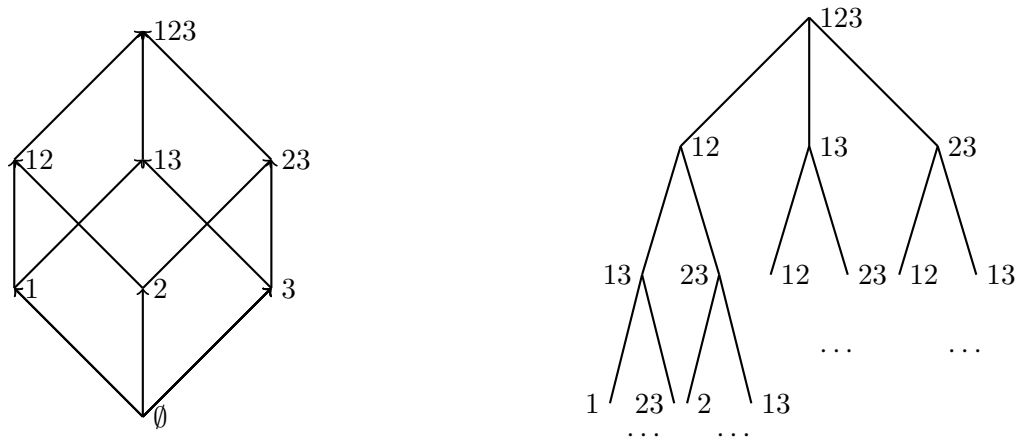

Figure 1: Case $n = 3$. Left: representation of the poset $(2^N, \subseteq)$. Right: Dendrogram of the maximal elements when running the procedure for generating all linear extensions using the DFS algorithm.

Input: n, k integers.
% k is the number of all linear extensions and $n = |N|$.
Output: k generated capacities on 2^N
% *AllLinear* is an empty array which will contain all linear extensions
1: $count \leftarrow 0$
% P is an array containing the poset $2^N \setminus \{\varnothing, N\}$
2: **All-linear-extension**$(P, AllLinear, count)$
3: **repeat** k **times**
4: Select uniformly one linear extension of *AllLinear*
5: Generate uniformly $2^n - 2$ numbers between 0 and 1, sort them from smallest to largest, and assign them to the selected linear extension
end repeat

Algorithm 2

All-linear-extension$(P, AllLinearExtensions, count)$

%*AllLinearExtensions* stores all linear extensions of P and *count* stores the number of linear extensions
Input: an array P containing a poset of size n, an array *AllLinearExtensions* and *count*
Output: All linear extensions of poset P
6: **If** $|P| = 1$ **then**
% When the bottom of dendrogram is reached, add an empty linear

extension to $AllLinearExtensions$.
7: Append a zeros array of size n to $AllLinearExtensions$
8: $AllLinearExtensions[count-1][n-1] \leftarrow P[0]$
9: $count \leftarrow count + 1$
end if
10: **For** i in $\mathsf{Max}(P)$ **do**
11: Remove i from P
% recursion algorithm
12: **All-linear-extension**$(P, AllLinearExtensions, count)$
13: $AllLinearExtensions[count-1]$[size of P] $\leftarrow$ i
14: Re-insert i to the end of poset P
end for

When $n > 4$, approximate methods have to be used, either generating randomly linear extensions, or based on other principles. In the first category, two methods generate linear extensions with the exact distribution. The first one is the Markov Chain method [14] (see description in Appendix), and the second one is the method of De Loof et al. [19]. The latter uses the lattice of ideals $I(P)$ of the poset P under consideration, and is based on the fact that a linear extension corresponds to a path in $I(P)$. This permits to count the number of linear extensions of P and of any subposet of P, which in turn yields the exact probabilities of choosing the next element when constructing a linear extension. The method is applicable till $n = 5$ (see [4]), because beyond the size of $I(2^N)$ becomes too large. We mention also the 2-layer approximation method [10], yielding approximate distributions. In the second category, we find the *Random Node generator (RNG) algorithm* introduced by T. C. Havens and A. J. Pinar in [12]. The core idea of this approach is to randomly select one element $S \in 2^N$ among all elements and then draw it with a uniform law between the maximum and minimum values allowed by the monotonicity constraints. This operation is repeated until all elements in 2^N have assigned values.

The most significant advantage of this method is its low complexity and fast running speed. However, theoretically, the capacities generated by it are not uniform, because firstly the range of values for $\mu(S)$ is highly dependent on the rank in which the element S is selected, and secondly the exact distribution of $\mu(S)$ is far from being a uniform distribution. Therefore, this capacity generator has a lot of theoretical undesirabilities.

As an illustration, we compare the performance of the RNG with ECG. The following figures show the distribution of $\mu(S)$ generated by the RNG and ECG.

From Figures 2 and 3, we notice that the discrepancy between the distribution of these two groups of μ is significant, and thus we may conclude that the uniformity of the capacity obtained by the Random-Node generator is not satisfactory. In the next subsections, we study the theoretical distribution of $\mu(S)$ and propose an improvement of the Random-Node generator.

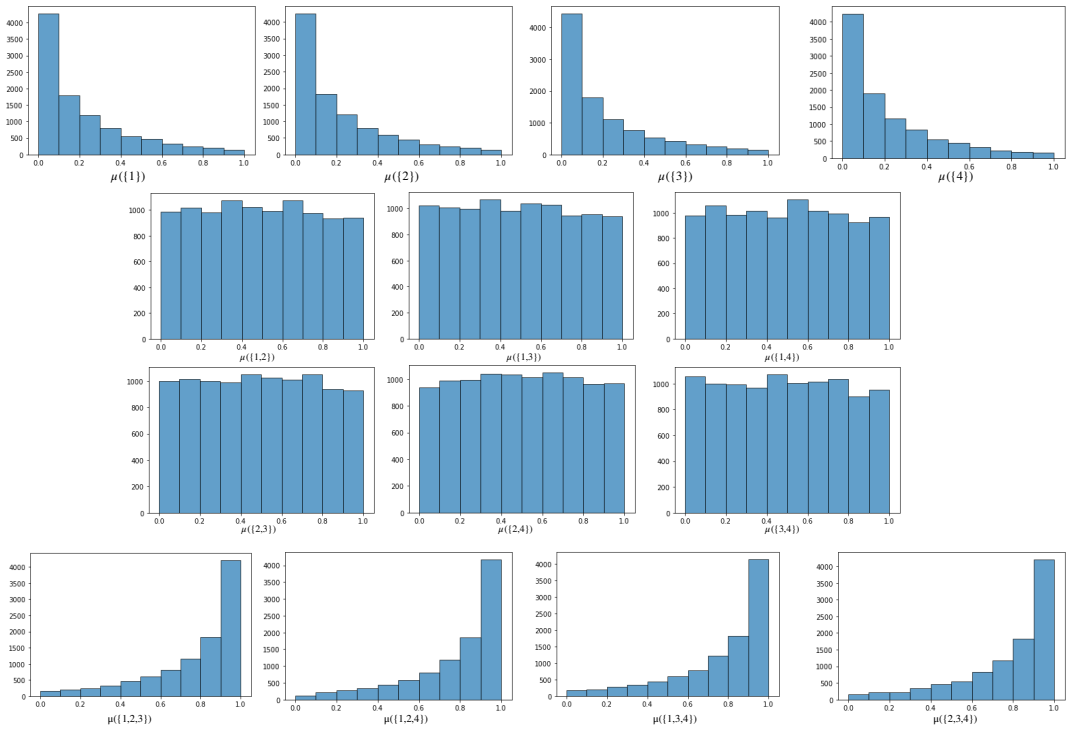

Figure 2: Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by RNG (compare with Fig. 3 where the exact generator has been used).

3.2 Theoretical distribution of μ

The main idea for improving the random node generator algorithm is to use a more realistic probabilistic distribution on the generation of the capacity of the current subset S. Let us first describe the probability distribution of such a term.

To this end, let us consider a set of i.i.d random variables $\mu_1, \mu_2, \ldots, \mu_m$ that follow the uniform law between 0 and 1. We sort the μ_i into the order statistics $\mu_{(1)} \leq \mu_{(2)} \cdots \leq \mu_{(m)}$. Then $\mu_{(k)}$ follows the Beta distribution $\mu_{(k)} \sim \text{Beta}(k, m-k+1)$. If we take $\alpha = k, \beta = m-k+1$, then the formula for the density of $\mu_{(k)}$ is as follows:

$$f_{\mu_{(k)}}(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1},$$

where $\Gamma(\alpha) = \int_0^\infty t^{\alpha-1}e^{-t}dt$, with $\alpha > 0$.

In order to apply this result to capacities, we need to know the rank of $\mu(S)$ within all terms of a capacity. We denote by $\mathcal{R}\hbar(S)$ the rank of element S ($S \in 2^N$) in a linear extension of poset $(2^N, \subseteq)$. Each element of 2^N has a rank in each linear extension corresponding to the poset, among them $\varnothing$ is always located at the minimal rank, i.e., $\mathcal{R}\hbar(\varnothing) = 0$ and N is always located at the maximal rank, i.e., $\mathcal{R}\hbar(N) = 2^n - 1$.

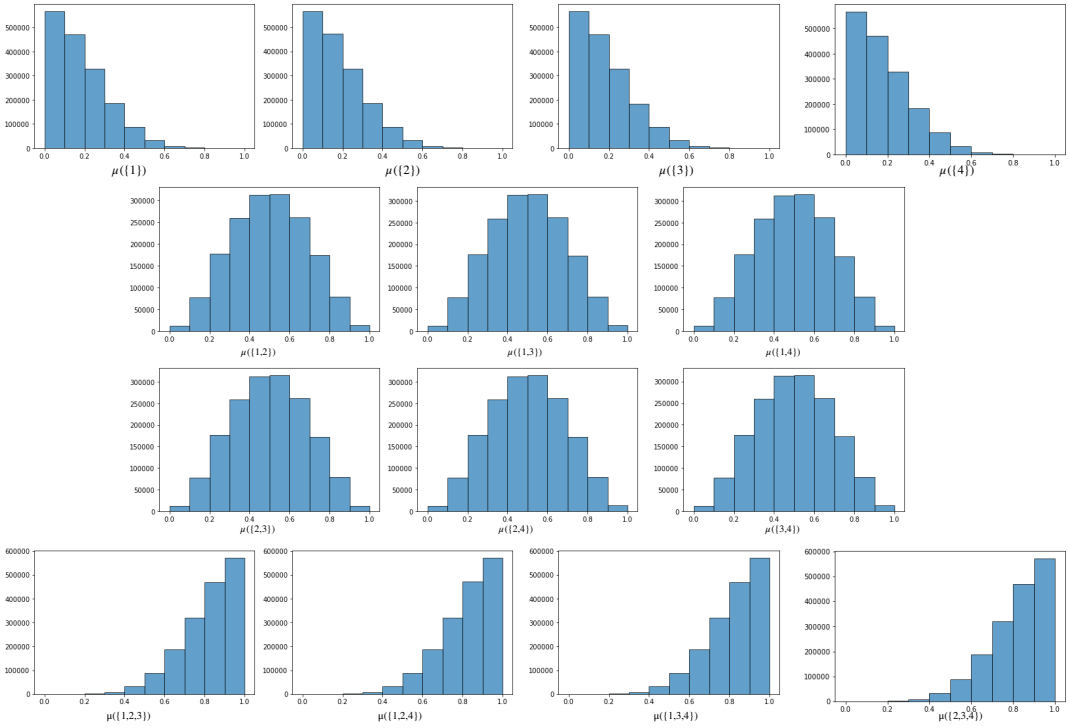

Figure 3: Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by ECG.

Then the cumulative distribution function of $\mu(S)$, i.e. $\mathbb{P}(\mu(S) \leq x)$ for $0 < x < 1$, considers the beta distribution over all possible rankings of $\mu(S)$

$$F_{\mu(S)}(x) = \mathbb{P}(\mu(S) \leq x) = \sum_{i=\min(\mathcal{R}k(S))}^{\max(\mathcal{R}k(S))} \mathbb{P}(\mu(S) \leq x | \mathcal{R}k(S) = i) \times \mathbb{P}(\mathcal{R}k(S) = i)$$

$$= \sum_{i=\min(\mathcal{R}k(S))}^{\max(\mathcal{R}k(S))} \mathbb{P}(\mu_{(i)} \leq x) \times \mathbb{P}(\mathcal{R}k(S) = i)$$

$$= \sum_{i=\min(\mathcal{R}k(S))}^{\max(\mathcal{R}k(S))} F_{\mu_{(i)}}(x) \times \mathbb{P}(\mathcal{R}k(S) = i),$$

with $F_{\mu_{(i)}}(x)$ the cumulative distribution function of $\text{Beta}(i, 2^n - 1 - i)$, $\min(\mathcal{R}k(S))$ the smallest possible ranking of $\mu(S)$ and $\max(\mathcal{R}k(S))$ the largest possible ranking of $\mu(S)$. These bounds on the ranking of $\mu(S)$ are simply obtained by the monotonicity condition, counting the minimal number of terms ranked before and after $\mu(S)$. We obtain

$$\min(\mathcal{R}k(S)) = |\{T \subseteq S \, , T \neq \emptyset\}| = 2^{|S|} - 1$$
$$\max(\mathcal{R}k(S)) = 2^n - |\{T \supseteq S \ , \ T \subseteq N\}| = 2^n - 2^{|N \setminus S|}.$$

The density of $\mu(S)$ is thus:

$$f_{\mu(S)}(x) = \sum_{i=min(\mathcal{R}k(S))}^{max(\mathcal{R}k(S))} f_{\mu_{(i)}}(x) \times \mathbb{P}(\mathcal{R}k(S) = i) \tag{4}$$

with $\mu_{(i)} \sim \text{Beta}(i, 2^n - 1 - i)$.

Density (4) is correct when $\mu(S)$ is not constrained by other terms of the capacity. When we use the RNG to generate a capacity, we should adjust the above distribution due to its monotonicity. Supposing we have already generated the elements $S_1, \ldots, S_p$ with the values $\mu(S_1) = a_1, \ldots, \mu(S_p) = a_p$, we wish to draw the distribution of $\mu(S)$ for a new subset S. Compared to (4), the knowledge of $a_1, \ldots, a_p$ provides constraints on both the numerical value of $\mu(S)$ and also its ranking. Following the monotonicity conditions, we first note that the value of $\mu(S)$ shall belong to interval $[\mathsf{Min}_{\mathsf{p}}\mu(S), \mathsf{Max}_{\mathsf{p}}\mu(S)]$ where

$$\mathsf{Min}_{\mathsf{p}}\mu(S) = \max_{j \in \{1,\ldots,p\} \, , \, S_j \subseteq S} a_j \quad \text{and} \quad \mathsf{Max}_{\mathsf{p}}\mu(S) = \min_{j \in \{1,\ldots,p\} \, , \, S_j \supseteq S} a_j.$$

Moreover, as illustrated by the following example, the smallest and largest possible rankings of $\mu(S)$ are also constrained by $a_1, \ldots, a_p$.

Example 1. *Assume that we have already generated the following terms* $\mu(\{1,2\}) = 0.1$, $\mu(\{1,3\}) = 0.2$ *and* $\mu(\{4,5\}) = 0.3$, *and consider now* $S = \{1,4,5\}$ *with* $N = \{1,2,3,4,5\}$. *Subset* $\{1,2\}$ *and all its subsets are thus ranked before* $\{4,5\}$. *The same*

holds for $\{1,3\}$. *In total, the subsets that are necessarily ranked before* S *are the following:* $\{1\},\{2\},\{1,2\},\{3\},\{1,3\},\{4\},\{5\},\{4,5\},\{1,4\},\{1,5\}$. *Hence* S *has rank at least* 11. ■

Generalizing the previous example,

$$\underline{\mathcal{S}}_p(S) = \{S_j \ , \ j \in \{1,\dots,p\} \text{ s.t. } \exists i \in \{1,\dots,p\} \ , \ S_i \subseteq S \text{ and } a_j \leq a_i\} \cup \{S\}$$

is the set of already generated subsets that are necessarily ranked before S (including S), and

$$\overline{\mathcal{S}}_p(S) = \{S_j \ , \ j \in \{1,\dots,p\} \text{ s.t. } \exists i \in \{1,\dots,p\} \ , \ S_i \supseteq S \text{ and } a_j \geq a_i\} \cup \{S\}$$

is the set of already generated subsets that are necessarily ranked after S (including S). The smallest possible ranking $\mathsf{Min_p}\mathcal{R}\hbar(S)$ of S is thus given by the number of subsets of $\underline{\mathcal{S}}_p(S)$. It is not simply the sum of the subsets of the elements of $\underline{\mathcal{S}}_p(S)$ as there are common subsets. In Ex. 1, subset $\{1\}$ is a subset of $\{1,2\}$, $\{1,3\}$ and $\{1,4,5\}$, and it shall not be counted three times. To this end, we use the Poincaré sieve formula. This formula provides the number of elements of the union of an arbitrary number of sets:

$$|\cup_{i=1}^q A_i| = \sum_{k=1}^{q} \left((-1)^{k-1} \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq q} |A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k}| \right).$$

We apply this formula to $A_j = 2^{\underline{S}_j} \setminus \{\emptyset\}$, where $\underline{\mathcal{S}}_p(S) := \{\underline{S}_1, \dots, \underline{S}_q\}$. As $A_{i_1} \cap \cdots \cap A_{i_k} = 2^{\underline{S}_{i_1} \cap \cdots \cap \underline{S}_{i_k}} \setminus \{\emptyset\}$, we obtain

$$\begin{aligned} \mathsf{Min_p}\mathcal{R}\hbar(S) &= \left|\{T \subseteq \underline{S}_j, \ T \neq \emptyset \text{ and } j \in \{1,\dots,q\}\right| \\ &= \sum_{k=1}^{q} \left((-1)^{k-1} \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq q} \left(2^{\left|\underline{S}_{i_1} \cap \underline{S}_{i_2} \cap \cdots \cap \underline{S}_{i_k}\right|} - 1 \right) \right). \end{aligned} \tag{5}$$

Example 2 (Ex. 1 continued)**.** *We obtain* $\mathsf{Min_p}\mu(\{1,4,5\}) = 0.3$. *Moreover, we have* $\underline{\mathcal{S}}_p(\{1,4,5\}) = \big\{\{1,2\},\{1,3\},\{4,5\},\{1,4,5\}\big\}$. *Applying (5), the smallest possible ranking of* $\{1,4,5\}$ *is*

$$\begin{aligned} &\left(2^{|\{1,2\}|}-1\right) + \left(2^{|\{1,3\}|}-1\right) + \left(2^{|\{4,5\}|}-1\right) + \left(2^{|\{1,4,5\}|}-1\right) \\ &- \left(2^{|\{1\}|}-1\right) - \left(2^{|\{1\}|}-1\right) - \left(2^{|\{1\}|}-1\right) - \left(2^{|\{4,5\}|}-1\right) + \left(2^{|\{1\}|}-1\right) \\ &= 3+3+3+7-1-1-1-3+1 = 11. \end{aligned}$$

Hence we recover that S *has rank at least* 11. ■

Likewise, the largest possible ranking $\mathsf{Max_p}\mathcal{R}\hbar(S)$ of S is given by

$$2^n - 1 - \left|\{T \supseteq \overline{S}_j,\ T \neq N \text{ and } j \in \{1,\ldots,q'\}\right|,$$

where $\overline{S}_p(S) := \{\overline{S}_1,\ldots,\overline{S}_{q'}\}$. Applying the Poincaré sieve formula to $A_j = \{T \subseteq \overline{S}_j\ ,\ T \neq N\}$, we obtain $|A_{i_1} \cap \cdots \cap A_{i_k}| = |\{T \supseteq \overline{S}_{i_1},\ldots,\overline{S}_{i_k}\ ,\ T \neq N\}| = |\{T \supseteq \overline{S}_{i_1} \cup \cdots \cup \overline{S}_{i_k}\ ,\ T \neq N\}| = 2^{|N\setminus(\overline{S}_{i_1}\cup\cdots\cup\overline{S}_{i_k})|} - 1$ and

$$\begin{aligned}
&\mathsf{Max_p}\mathcal{R}\hbar(S) = 2^n - 1 - \left|\{T \supseteq \overline{S}_j,\ T \neq N \text{ and } j \in \{1,\ldots,q'\}\right| \\
&= 2^n - 1 - \sum_{k=1}^{q'}\left((-1)^{k-1}\sum_{1\leq i_1<i_2<\cdots<i_k\leq q'}\left(2^{|N\setminus(\overline{S}_{i_1}\cup\cdots\cup\overline{S}_{i_k})|}-1\right)\right). \qquad (6)
\end{aligned}$$

Example 3. *Assume that we have already generated the following terms* $\mu(\{1,2,3\}) = 0.9$, $\mu(\{1,3,4\}) = 0.8$ *and* $\mu(\{1,2,4,5\}) = 0.7$, *and consider now* $S = \{1,2,5\}$ *with* $N = \{1,2,3,4,5\}$. *We obtain* $\mathsf{Max_p}\mu(\{1,2,5\}) = 0.7$. *Moreover, we have* $\overline{S}_p(\{1,2,5\}) = \big\{\{1,2,3\},\{1,3,4\},\{1,2,4,5\},\{1,2,5\}\big\}$. *The subsets (excluding* N*) ranked after* $\{1,2,5\}$ *are* $\{1,2,5\},\{1,2,3,5\},\{1,2,4,5\},\{1,2,3\},\{1,2,3,4\},\{1,3,4\},\ \{1,3,4,5\}$. *We obtain* 7 *subsets.*

Applying (6), the largest possible ranking of $\{1,2,5\}$ *is*

$$\begin{aligned}
&2^5 - 1 - \left(2^{|\{3,4\}|}-1\right) - \left(2^{|\{4,5\}|}-1\right) - \left(2^{|\{2,5\}|}-1\right) - \left(2^{|\{3\}|}-1\right) \\
&+\left(2^{|\{5\}|}-1\right) + \left(2^{|\{4\}|}-1\right) - \left(2^{|\{3\}|}-1\right) \\
&= 2^5 - 1 - 3 - 3 - 3 - 1 + 3 = 2^n - 1 - 7 = 24
\end{aligned}$$

■

Summarizing, the distribution of $\mu(S)$ becomes a conditional distribution:

$$\begin{aligned}
&\mathbb{P}(\mu(S) \leq x | \mu(S_1) = a_1,\ldots,\mu(S_p) = a_p) \qquad (7) \\
&= \sum_{i=min_p\mathcal{R}\hbar(S)}^{max_p\mathcal{R}\hbar(S)} \mathbb{P}(\mathcal{R}\hbar(S) = i | \mu(S_1) = a_1,\ldots,\mu(S_p) = a_p) \\
&\qquad \times \mathbb{P}(\mu(S) = \mu_{(i)} \leq x | \mu(S_1) = a_1,\ldots,\mu(S_p) = a_p)
\end{aligned}$$

with

$$\begin{aligned}
&\mathbb{P}(\mathcal{R}\hbar(S) = i | \mu(S_1) = a_1,\ldots,\mu(S_p) = a_p) \\
&\approx \mathbb{P}(\mathcal{R}\hbar(S) = i | \mathsf{Min_p}\mathcal{R}\hbar(S) \leq \mathcal{R}\hbar(S) \leq \mathsf{Max_p}\mathcal{R}\hbar(S)) \\
&= \begin{cases} \frac{\mathbb{P}(\mathcal{R}\hbar(S)=i)}{\sum_{j=min_p\mathcal{R}\hbar(S)}^{max_p\mathcal{R}\hbar(S)}\mathbb{P}(\mathcal{R}\hbar(S)=j)} & \text{if } \mathsf{Min_p}\mathcal{R}\hbar(S) \leq i \leq \mathsf{Max_p}\mathcal{R}\hbar(S) \\ 0 \quad \text{else} \end{cases} \qquad (8)
\end{aligned}$$

and

$$\begin{aligned}&\mathbb{P}(\mu(S) = \mu_{(i)} \leq x | \mu(S_1) = a_1, \ldots, \mu(S_p) = a_p)\\ &= \mathbb{P}(\mu_{(i)} \leq x | \mathsf{Min_p}\mu(S) \leq \mu_{(i)} \leq \mathsf{Max_p}\mu(S)).\end{aligned} \tag{9}$$

In the approximation leading to (8), we assume that the probability that $\mathcal{R}\hbar(S) = i$ does not depend on the numerical values of $a_1, \ldots, a_p$. We assume that it is proportionnal to $\mathbb{P}(\mathcal{R}\hbar(S) = i)$ (the probability that $\mathcal{R}\hbar(S) = i$ in the unconstrained case, i.e. when there are no constraints on terms of the capacity except the monotonicity conditions) when i lies in the admissible range, and is equal to 0 otherwise.

3.3 The improved random node generator

Thanks to the previous considerations and Equations (7), (8) and (9), we are in a position to propose an improvement of the random node generator, which we call IRNG.

As explained, our improvement consists in replacing the uniform distribution of $\mu(S)$ in the interval $[\mathsf{Min_p}\mu(S), \mathsf{Max_p}\mu(S)]$ by the distribution given by (7), computed through (8) and (9).

According to Equation (9), when we assign a value to $\mu(S)$, it should be between $\mathsf{Min_p}\mu(S)$ and $\mathsf{Max_p}\mu(S)$. If this is not satisfied, we need to reject it and reassign a new value to $\mu(S)$.

As for Equation (8), it is necessary to know the probability $\mathbb{P}(\mathcal{R}\hbar(S) = i)$ for a given subset S to be ranked at ith position in a linear extension. This probability is stored in array *probability* (where $probability[S][i] = \mathbb{P}(\mathcal{R}\hbar(S) = i)$) in the following algorithm. We denote by $CDF^S_{prob}(i) = \sum_{j=0}^{i} probability[S][j]$ the cumulative distribution function of array *probability*. As the set of linear extensions is not practically reachable beyond $n = 5$ and not known beyond $n = 8$, no practical expression of this probability can be obtained, and it must be estimated. Therefore, the critical issue for the precision of the IRNG algorithm is how to get these probabilities. Our proposition is to use off line some well-performing method to generate randomly in a uniform way linear extensions of $(2^N \setminus \{\varnothing, N\}, \subseteq)$, like the Markov chain method [14], generating a sufficient number of linear extensions from which $\mathbb{P}(\mathcal{R}\hbar(S) = i)$ could be estimated, for every subset S and every rank i [1]. Once we have obtained these probabilities, we store them in a file[2] so that they can be used repeatedly.

[1] We compared the impact on IRNG for n = 4 by varying the quality of ranking probabilities. When employing the exact method to determine the ranking probabilities, the sum of Kullback-Leibler divergence (KLD) is around 0.40. When employing the Markov chain method with a high number of interations (T= 1000), the sum of KLD is 0.42, which is very close to the case of exact probabilities. Now, with a low number of iterations (T = 50) insufficient for a good convergence of the Markov chain, we obtain for the sum of KLD the value of 1.61. Therefore, it is evident that the quality of ranking probabilities has a significant impact on the outcomes of IRNG.

[2] The reader can find as supplementary material of this article the files of probabilities for $n = 3$ and $n = 4$.

Algorithm 3

Improved-Random-Node-generator (IRNG)$(P, probability)$

Input: a poset P of $2^N \backslash \{\varnothing, N\}$, a two dimensional array named $probability[S][j]$ containing the probability of element (subset) $S \in 2^N \setminus \{\varnothing, N\}$ to be at rank j.
Output: capacity μ in $\mathcal{C}(N)$ generated with approximation method

1: AssignedElement,AssignedValue $\leftarrow$[ ],[ ]
2: $\mu \leftarrow$ a zero array of size $2^n - 2$
%AssignedElement and AssignedValue store the elements $S_1, \ldots, S_p$
%and element's value $a_1, \ldots, _p$ that have been already assigned
3: $\mathcal{L} \leftarrow$ an array of elements of $2^N \setminus \{\varnothing, N\}$ in random order
4: $p \leftarrow 0$
5: **for** S in $\mathcal{L}$ **do**
6: Compute $\mathsf{Min_p}\mu([S]), \mathsf{Max_p}\mu([S])$ and $\mathsf{Min_p}\mathcal{R}\hbar(S)$, $\mathsf{Max_p}\mathcal{R}\hbar(S)$
% $\mathsf{Min_p}\mathcal{R}\hbar(S)$, $\mathsf{Max_p}\mathcal{R}\hbar(S)$ the ranking restrictions of S
% and $\mathsf{Min_p}\mu([S]), \mathsf{Max_p}\mu([S])$ the minimum and maximum value of $\mu([S])$
7: $beta \leftarrow 0$
8: $\Pr_{min} \leftarrow CDF^S_{prob}(\mathsf{Min_p}\mathcal{R}\hbar(S)))$
9: $\Pr_{max} \leftarrow CDF^S_{prob}(\mathsf{Max_p}\mathcal{R}\hbar(S)))$
10: **While** $beta \geq \mathsf{Max_p}\mu([S])$ **or** $beta \leq \mathsf{Min_p}\mu([S])$ **do**
% Capacity should obey monotonicity
11: $r \sim U([0,1])$
12: $r \leftarrow \Pr_{min} + (\Pr_{max} - \Pr_{min}) * r$
13: $Rank \leftarrow \mathsf{Min_p}\mathcal{R}\hbar(S))$
14: $\Pr \leftarrow \Pr_{min}$
15: **While** $r > \Pr$ **do**
16: $\Pr \leftarrow CDF^S_{prob}(Rank)$
17: $Rank \leftarrow Rank + 1$
end while
18: $beta \sim \text{Beta}(Rank, 2^n - 1 - Rank)$.
end while
19: $\mu[S] \leftarrow beta$
20: Append $\mu[S]$ to Assignedvalue
21: Append S to AssignedElement
22: $p \leftarrow p + 1$
end for

Let us analyze the computational complexity of one run to IRNG. The $2^n - 2$ subsets are ordered in array $\mathcal{L}$. In $l.5$, we sweep these elements with an index p from $p = 1$ to $p = 2^n - 2$. At iteration p $(l.6 - 22)$, the complexity is given by the successive steps:

- $l.6$: the computation of $\mathsf{Min_p}\mu([S])$ and $\mathsf{Max_p}\mu([S])$ requires p operations;
- $l.6$: the computation of $\mathsf{Min_p}\mathcal{R}\hbar(S)$ and $\mathsf{Max_p}\mathcal{R}\hbar(S)$ requires $2^q + 2^{q'} \leq 2^p$ operations (see (5) and (6));
- $l.8-9$: the computation of Pr_{min} and Pr_{max} requires at most 2^n operations;
- Complexity of the While loop in $l.10$: For $Rank \in \{\mathsf{Min_p}\mathcal{R}\hbar(S), \ldots, \mathsf{Max_p}\mathcal{R}\hbar(S)\}$ fixed, we generate a number *beta* according to distribution $\mathrm{Beta}(Rank, 2^n - 1 - Rank)$. We need to rerun the While loop if *beta* does not belong to interval $[\mathsf{Min_p}\mu([S]), \mathsf{Max_p}\mu([S])]$. Let $CDF_{\mathrm{Beta}}^{Rank, 2^n-1-Rank}(\cdot)$ be the CDF of the beta distribution. Then the probability of rejecting *beta* under *Rank* is

$$\begin{aligned} P_{\mathrm{reject}}(Rank) =& 1 - CDF_{\mathrm{Beta}}^{Rank, 2^n-1-Rank}(\mathsf{Max_p}\mu([S])) \\ &+ CDF_{\mathrm{Beta}}^{Rank, 2^n-1-Rank}(\mathsf{Min_p}\mu([S])). \end{aligned}$$

  Hence the probability of rejection in a While loop is

$$P_{\mathrm{reject}} = \sum_{Rank=\mathsf{Min_p}\mathcal{R}\hbar(S)}^{\mathsf{Max_p}\mathcal{R}\hbar(S)} \widehat{probability}[S][Rank] \times P_{\mathrm{reject}}(Rank),$$

  where

$$\widehat{probability}[S][Rank] = \frac{probability[S][Rank]}{\sum_{k=\mathsf{Min_p}\mathcal{R}\hbar(S)}^{\mathsf{Max_p}\mathcal{R}\hbar(S)} probability[S][k]}.$$

  We conclude that the probability that one needs to run ℓ times the While loop to have an acceptance (i.e. the While loop rejects $\ell - 1$ times the generated *beta*, and accepts it juste after) is

$$P_{\mathrm{reject}}(\ell) = {P_{\mathrm{reject}}}^{\ell-1} \times (1 - P_{\mathrm{reject}}). \tag{10}$$

  This number obviously depends on the probability of the beta distribution to be outside interval $[\mathsf{Min_p}\mu([S]), \mathsf{Max_p}\mu([S])]$. We see that $P_{\mathrm{reject}}(\ell)$ decreases exponentially to zero with ℓ.

  The While loop in $l.15$ is run at most 2^n times. Then the complexity of $l.10-18$ is $\ell \times 2^n$, where ℓ is the number of times to converge to in the While loop at $l.10$.

In total, the complexity of one run of IRNG is $O(2^n)$. The main uncertainty in the computation time is the number of times M the While loop in $l.10$ is run. In the worst case, it could be large if interval $[\mathsf{Min_p}\mu([S]), \mathsf{Max_p}\mu([S])]$ is very small and *Rank* is not well adapted to this interval. This situation occurs with a low probability.

Figure 4: Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by IRNG (compare with Fig. 3 where the exact generator has been used).

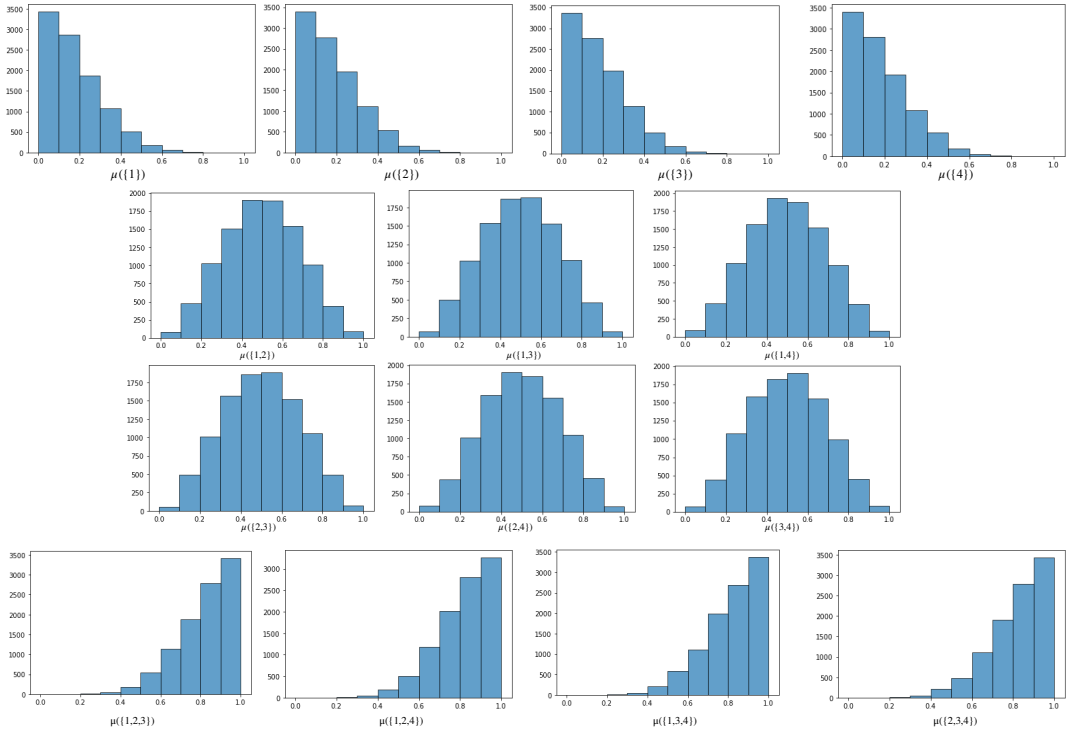

Figure 5: Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by the Markov chain generator with the number of iterations (T) equal to 1000 (compare with Fig. 3).

4 Experimental results for IRNG

We compare the performance of the IRNG with the RNG and Markov Chain generator. When $n = 4$, we apply ECG to obtain $\mathbb{P}(\mathcal{R}k(S) = i)$, while the Markov chain generator is applied when $n = 5$. Figure 4 shows the distribution of $\mu(S)$ generated by the IRNG for $n = 4$.

From Figure 4, we notice that the distribution of μ generated by the IRNG is much closer to the exact distribution than the one generated with the classical RNG (Fig. 2), and Figure 5 shows the distribution of μ generated by the Markov Chain generator.

Next, we further compare their performance by calculating the Kullback-Leibler divergence (also called Relative entropy) between the distributions of $\mu(S)$ obtained by the exact generator and those obtained by the considered generators, which could be

used to estimate the similarity of two distributions.
Recall the definition of Kullback-Leibler divergence (KLD for short):

$$\mathbb{D}_{KL}(p||q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}$$

with p and q two discrete probability distributions defined on the same probability space $\mathcal{X}$.

In our experiments, we need to compare the distribution of $\mu(S)$ generated by the considered generators with the exact distribution of $\mu(S)$. We replace q by the exact distribution of $\mu(S)$ and p by the distribution of $\mu(S)$ obtained by one of these three generators and then compare their value. The smaller the value, the higher the similarity with the exact distribution (shown in Table 1).

Table 2 shows the CPU time of different capacity generators (we have used Python implementations of the algorithms described above and have conducted the experiments on a 3.2 GHz PC with 16 GB of RAM). For the execution time of IRNG, the time required to compute the probabilities in Equation (8) is not taken into account, as they are computed once for all off line.

capacity generator	$\mu(\{1\})$	$\mu(\{2\})$	$\mu(\{3\})$	$\mu(\{4\})$
RNG	0.4220	0.3651	0.3708	0.3947
IRNG	0.0332	0.0373	0.0343	0.0351
Markov Chain	0.0115	0.0109	0.0097	0.0073

capacity generator	$\mu(\{1,2\})$	$\mu(\{1,3\})$	$\mu(\{1,4\})$	$\mu(\{2,3\})$	$\mu(\{2,4\})$	$\mu(\{3,4\})$
RNG	0.6677	0.6322	0.7401	0.6522	0.6836	0.6375
IRNG	0.0180	0.0237	0.0232	0.0267	0.0268	0.0306
Markov Chain	0.0093	0.0090	0.0110	0.0108	0.0090	0.0102

capacity generator	$\mu(\{1,2,3\})$	$\mu(\{1,2,4\})$	$\mu(\{1,3,4\})$	$\mu(\{2,3,4\})$
RNG	0.3985	0.3818	0.3691	0.3701
IRNG	0.0265	0.0317	0.0284	0.0267
Markov Chain	0.0089	0.0072	0.0081	0.0094

Table 1: Kullback-Leibler divergence between the histograms produced by the considered generators and those produced by the exact generator

From Table 1, we compute the sum of the KLD for $\mu(S)$ ($\forall S \in 2^N \setminus \{\varnothing, N\}$) for each generator. We obtain that the value for RNG is 7.086, for IRNG is 0.40 and for Markov Chain is 0.132. As can be seen from these results, IRNG is approximately twenty times

better than the original RNG method, and yields a sum of KLD of the same order of magnitude of that of the Markov Chain method.

Compared to the RNG, the distribution of μ obtained from the IRNG is considerably improved and does not differ significantly from the distribution obtained with the Markov chain generator.

Method	four elements' capacity	five elements' capacity
RNG	0.425	1.130
IRNG	2.142	16.135
Markov Chain Generator	17.7	161.33

Table 2: Comparison of CPU time (measured in second) for generating 10,000 capacities. Among them, the number of iterations (T) used for the Markov Chain Generator is 1,000 when $n = 4$, and 9,000 when $n = 5$.

Unlike RNG, IRNG needs to compute $\mathsf{Min_p}\mathcal{R}\hbar(S)$ and $\mathsf{Max_p}\mathcal{R}\hbar(S)$ for each S. Therefore, IRNG is theoretically more complex than RNG, and this difference is reflected in the computation time. However, from Table 2, this difference remains negligible in view of the time required by the Markov chain method, and it can be seen that IRNG is much faster than the Markov Chain Generator. This definitely shows the advantage of IRNG, whose performance is dramatically better than that of RNG. To further evaluate the performance and the computational time of IRNG and the Markov Chain generator, we systematically vary the number of iterations (T) of Markov Chain method and compute the sum of KLD for $n = 4$. The experimental results are summarized in Table 3.

T	100	150	160	170	180	190	200	250
Sum of KLD	0.85	0.55	0.51	0.47	0.44	0.41	0.38	0.28
Time	1.7s	2.6s	2.8s	3.0s	3.1s	3.3s	3.5s	4.3s

Table 3: Sum of KLD between the distribution of μ generated by Markov Chain method with different values of T and the exact distribution. Computation for 10,000 samples and $n = 4$

For the IRNG, the sum of KLD is approximately 0.40, and the computational time required is 2.1s. We observe that when the CPU times of both methods are equivalent ($T = 120$), the performance of IRNG surpasses that of the Markov Chain method. Indeed, with $T = 120$, the sum of KLD is around 0.72. On the other hand, if the same accuracy is required, the MC method needs a computation time of 3.3s ($T = 190$), which is longer, but of the same order of magnitude.

For $n = 5$, we have computed the sum of KLD between each pair of distributions of $\mu(S)$ and $\mu(S')$ with $|S| = |S'|$, thus checking the symmetry property. Results are shown on Table 4. We observed that for IRNG, the sum of KLD is approximately 2.3, and the computational time is around 15.8s. Consequently, when considering the performance of both methods to be consistent, it is evident that the Markov Chain method requires

T	100	500	2000	4000	6000	7000	8000
Sum of KLD	152.3	58.4	9.6	3.3	2.4	2.3	2.4
Time	1.7s	8.9s	35.7s	71.4s	106.8s	124.5s	142.6

Table 4: Sum of KLD between each pair of distributions of $\mu(S)$ and $\mu(S')$ with $|S| = |S'|$ for a sample size of 10,000 ($n = 5$)

significantly more computational time than IRNG, namely 124.5s. Conversely, when aiming to maintain consistent time costs for both methods, the performance of the Markov Chain method is notably inferior to that of IRNG.

5 Capacity generators with preference information

In this section, we would like to add some additional restrictions to the ECG and IRNG, so that they could be applied in a multi-criteria decision problem based on Choquet integral preference model.

5.1 Statement of the problem

Consider a discrete multiple criteria decision problem with n criteria where the decision is based on a Choquet integral, as described in Section 2.1. The capacity is obtained from preference information provided by the DM, in terms of indifference statements I and strict preference statements P regarding some alternatives in a set of m alternatives denoted by A. We denote these preference information constraints as the system (S_R):

$$(S_R) \qquad \begin{cases} C_\mu(a) = C_\mu(b), \text{ with } (a,b) \in I \\ C_\mu(a) \geq C_\mu(b) + \epsilon, \text{ with } (a,b) \in P. \end{cases}$$

In the above system, we set the value of $\epsilon > 0$ to be sufficiently small to ensure that there is no equality implied by the system of inequalities.

The purpose of this section is to randomly generate a set of capacities compatible with system (S_R), in a uniform way.

The simplest method for generating capacities compatible with (S_R) is acceptance and rejection, i.e., accepting only those generated capacities which are compatible with the system (S_R). The ratio of accepted capacities among all capacities is called the *acceptance rate*. Although this method is easily implementable, it will fail to succeed in practice if the acceptance rate is too small. In order to study this issue more explicitly, we introduce the following notation:

- We denote the volume of the polytope consisting of the capacities compatible with (S_R) as V_{S_R}, and the volume of capacities on 2^N as $V_{\mathcal{C}(N)}$.
- We denote by $r_n^a(S_R)$ the acceptance rate of generating an n elements' capacity compatible with (S_R), i.e., $r_n^a(S_R) = \frac{V_{S_R}}{V_{\mathcal{C}(N)}}$.

Since it does not exist a closed-form formula giving V_{S_R}, we cannot obtain a general conclusion about $r_n^a(S_R)$. Nevertheless, through the following example, we illustrate that the value of $r_n^a(S_R)$ could be quite low.

Example 4. *Suppose there are three criteria* ($n = 3$) *and 6 alternatives* ($m = 6$)*, with utility value as shown in the following table.*

Table 5: 3 criteria alternatives

item	criterion 1	criterion 2	criterion 3
a^1	6	8	7
a^2	7	1	8
a^3	4	3	8
a^4	4	9	7
a^5	9	1	5
a^6	9	4	3

Suppose the DM has the following preference information: $P_1 = \{(a^5, a^2), (a^2, a^3), (a^6, a^4)\}$*, that is,* $a^5 \succ a^2 \succ a^3$*, and* $a^6 \succ a^4$*. The system of preference information becomes:*

$$(S_{R_1}) \qquad \begin{cases} C_\mu(a^5) \geq C_\mu(a^2) + \epsilon \\ C_\mu(a^2) \geq C_\mu(a^3) + \epsilon \\ C_\mu(a^6) \geq C_\mu(a^4) + \epsilon. \end{cases}$$

The volume of the capacities compatible with (S_{R_1}) *(computed by software VINCI* [3]*) is 6.50e-03, and the volume* $V_{\mathcal{C}}(N)$ *is 6.67e-02 for* $n = 3$*. Hence, the acceptance rate of the above example is* 0.097*, which means that if we would like to generate 10,000 capacities compatible with* S_{R_1}*, we need ultimately to generate approximately* $103,092$ *general capacities, which shall take quite a lot of time.*

The above example shows that it is necessary to revise the algorithms of IRNG and ECG presented in section 3 in order to increase the acceptance rate by incorporating some constraints into the algorithms. However, the problem we encounter at this stage is that it would be too difficult to incorporate constraints like $4\mu(\{1\}) \geq \mu(\{3\}) + 2\mu(\{1,3\}) + \epsilon$ (first inequality of (S_{R_1})) in the algorithms, since the links among these three variables $\mu(\{1\}), \mu(\{3\}), \mu(\{1,3\})$ are too complex.

5.2 An approximation method

Our idea is basically to handle constraints between at most two variables, and limiting to very simple ones, in such a way that these rough constraints determine a polytope including the "true" polytope determined by the constraints (S_R), and as small as possible.

[3] https://www.multiprecision.org/vinci/home.html

We propose to restrict to two types of constraints, namely **constraint 1** and **constraint 2**.

$$(S_C) \qquad \begin{cases} \mu(S) \leq \mu(S') + \alpha, \text{ or } \mu(S) \geq \mu(S') + \alpha, \text{ with } S', S \in 2^N \text{ and } \alpha \in \mathbb{R} \textbf{ (constraint 1)} \\ \mu(S) \leq c, \mu(S) = c \text{ or } \mu(S) \geq c, \text{ with } c \text{ constant and } S \in 2^N \textbf{(constraint 2)} \end{cases}$$

As it will become clear later, algorithm IRNG is able to handle both types of constraints. However, the principle of ECG is to enumerate all linear extensions, and then randomly select one of them. The value of $\mu(S)$ is not involved in the process, therefore **constraint 2** would be impossible to add directly in the algorithm, while **constraint 1** could only be added directly when $\alpha = 0$.

We call generically (S_C) a set of constraints of this type. We explain below how to get a system (S_C) from a system (S_R), but we can already draw some important considerations. Denote by V_{S_C} the volume of the polytope determined by the system (S_C), supposing that the latter polytope contains the polytope of capacities determined by (S_R). As the two polytopes differ in general, we still need to apply the acceptance and rejection method to select only capacities compatible with (S_R). Therefore, there is an acceptance rate $r_n^a(S_C) = \frac{V_{S_R}}{V_{S_C}}$. If $\frac{V_{S_R}}{V_{S_C}}$ is much larger than $\frac{V_{S_R}}{V_{C(N)}}$ (acceptance rate $r_n^a(S_R)$ of the naive approach), then we have succeeded in reducing considerably the computation time of the algorithm.

We distinguish two types of alternative.

5.2.1 Case of binary alternatives.

Binary alternatives are special alternatives which utility values are limited to 0 and 1, and are denoted by a_B for $B \subseteq N$ – see Section 2.1 and Example 5 below. Binary alternatives are quite useful in practice. They represent efficient ways to elicit a capacity. They are central in particular in the MACBETH approach [3] and its extension to the Choquet integral [21, 22]. These alternatives are cognitively simple to represent for a DM as they are only good or bad with respect to all criteria.

By (3), the value of the Choquet integral for alternative a_B is simply

$$C_\mu(a_B) = \mu(B).$$

Therefore, the preferential information on binary alternatives induce only constraints of the type $\mu(B) \leq \mu(B')$, i.e., constraints of type **constraint 1** with $\alpha = 0$.

Example 5. *: Suppose there are three criteria and 5 binary alternatives shown in the following table. We use the Choquet integral to describe the preference of DM for these five alternatives. We obtain* $C_\mu(a_{\{1,3\}}) = \mu(\{1,3\})$, $C_\mu(a_{\{1,2\}}) = \mu(\{1,2\})$, $C_\mu(a_{\{2,3\}}) = \mu(\{2,3\})$ *and* $C_\mu(a_{\{2\}}) = \mu(\{2\})$. *Since* $\mu(\{2\}) \leq \mu(\{2,3\})$, $\mu(\{2\}) \leq \mu(\{1,2\})$, *alternative* $a_{\{2\}}$ *is dominated by alternatives* $a_{\{1,2\}}$ *and* $a_{\{2,3\}}$, *it could be ranked directly as the least favourite alternative. Suppose the DM prefers* $a_{\{1,3\}}$ *over* $a_{\{1,2\}}$, *then we have* $\mu(\{1,3\}) \geq \mu(\{1,2\})$, *which is of the type* ***constraint 1*** *with* $\alpha = 0$.

Table 6: 3 criteria binary alternatives

item	criteria 1	criteria 2	criteria 3
$a_{\{1,3\}}$	1	0	1
$a_{\{1,2\}}$	1	1	0
$a_{\{2,3\}}$	0	1	1
$a_{\{2\}}$	0	1	0
a_N	1	1	1

5.2.2 Case of arbitrary alternatives

Now let us consider the case where the utility value is not limited to 0 or 1. Firstly, observe that if two alternatives a, b are indifferent, we have $C_\mu(a) = C_\mu(b)$. Through this equation, one variable $\mu(S)$ ($S \in 2^N$) could be expressed in terms of the others in the equation and hence can be eliminated. Then, without loss of generality, one may consider that we have only strict preference information. In the same way, variable $\mu(N) = 1$ is eliminated.

The basic idea to get a system (S_C) of constraints which determines a smallest possible polytope is to get the minimal and maximal values of *every* non-eliminated variable $\mu(S)$ by linear programming, taking as constraints those given by (S_R) plus the monotonicity constraints, and similarly minimal values and maximal values of differences $\mu(S) - \mu(S')$, for every possible S, S'. For example, the corresponding linear program to obtain the minimal value of $\mu(S)$ for a fixed S reads:

$$(LP_S^{min}) \qquad \left\{ \begin{array}{lll} & \mathsf{Min}\ \mu(S) & \\ \text{subject to} & C_\mu(a) \geq C_\mu(b) + \epsilon & \text{for } (a,b) \in P \\ & \mu(S') \geq \mu(S''), & S' \supseteq S'' \end{array} \right.$$

with ϵ positive, $S', S'' \in 2^N \setminus \{\emptyset, N\}$. Similarly, for obtaining other inequalities in the system (S_C), we simply replace the linear objective function with $\mathsf{Max}\,\mu(S)$, $\mathsf{Min}\,[\mu(S) - \mu(S')]$ and $\mathsf{Max}\,[\mu(S) - \mu(S')]$ respectively, for all non-eliminated variables $\mu(S), \mu(S')$. This yields at most $(2^{n-1} - 1) * (2^n - 1)$ linear programs to solve, whose values permit to get the system (S_C). Denoting the minimum value of $\mu(S)$ by v_{min}^S, the maximum value of $\mu(S)$ by v_{max}^S, the minimum value of $\mu(S) - \mu(S')$ by $v_{min}^{S,S'}$ and the maximum value of $\mu(S) - \mu(S')$ by $v_{max}^{S,S'}$, we obtain

$$(S_C) \qquad \left\{ \begin{array}{ll} v_{min}^{S,S'} \leq \mu(S) - \mu(S') \leq v_{max}^{S,S'}, & \forall S, S' \in 2^N \setminus \{\emptyset, N\} \text{ and } S \neq S' \quad \textbf{(constraint 1)} \\ v_{min}^{S} \leq \mu(S) \leq v_{max}^{S}, & \forall S \in 2^N \setminus \{\emptyset, N\} \quad \textbf{(constraint 2)} \end{array} \right.$$

Example 4 continued: *We take $\epsilon = 0.01$ for the system (S_{R_1}), and then construct the family of linear programs to obtain the system (S_{C_1}). For example, for a given $S = \{1\}$, and $S' = \{2\}$, we specify $\mathsf{Min}[\mu(\{1\}) - \mu(\{2\})]$ as the linear objective function, and the linear program reads as follows :*

$$(LP^{min}_{(\{1\},\{2\})}) \quad \begin{cases} & \mathsf{Min}\,[\mu(\{1\}) - \mu(\{2\})] \\ \text{subject to} & C_\mu(a^i) \geq C_\mu(a^j) + 0.01 \quad \text{for } (a^i, a^j) \in P_1 \\ & \mu(S) \geq \mu(S') \qquad \forall S \supseteq S' \end{cases}$$

To obtain the solution of the above linear program, we use the solver GUROBI, and the result is $\mathsf{Min}[\mu(\{1\}) - \mu(\{2\})] = 0$*, implying that* $\mu\{1\} - \mu\{2\} \geq 0$*. Then we replace the objective function to obtain the remaining linear programs. After solving 42 linear programs, we obtain the following system of inequalities as the solution:*

$$(S_{C_1}) \quad \begin{cases} 0.2 \leq \mu\{1\} \leq 1; \quad 0 \leq \mu\{2\} \leq 1; \quad 0 \leq \mu\{3\} \leq 1 \\ 0.2 \leq \mu\{1,2\} \leq 1; \quad 0.4 \leq \mu\{1,3\} \leq 1; \quad 0 \leq \mu\{2,3\} \leq 1 \\ 0 \leq \mu\{1\} - \mu\{2\} \leq 1; \quad -0.25 \leq \mu\{1\} - \mu\{3\} \leq 1; \quad -0.8 \leq \mu\{1\} - \mu\{1,2\} \leq 0 \\ -0.5 \leq \mu\{1\} - \mu\{1,3\} \leq 0; \quad -0.4 \leq \mu\{1\} - \mu\{2,3\} \leq 1; \quad -1 \leq \mu\{2\} - \mu\{3\} \leq 1 \\ -1 \leq \mu\{2\} - \mu\{1,2\} \leq 0; \quad -1 \leq \mu\{2\} - \mu\{1,3\} \leq 0; \quad -1 \leq \mu\{2\} - \mu\{2,3\} \leq 0 \\ -1 \leq \mu\{3\} - \mu\{1,2\} \leq 0.25; \quad -1 \leq \mu\{3\} - \mu\{1,3\} \leq 0; \quad -1 \leq \mu\{3\} - \mu\{2,3\} \leq 0 \\ -0.5 \leq \mu\{1,2\} - \mu\{1,3\} \leq 0.6; -0.33 \leq \mu\{1,2\} - \mu\{2,3\} \leq 1.0; -0.4 \leq \mu\{1,3\} - \mu\{2,3\} \leq 1.0 \end{cases}$$

The numbers marked in blue are obtained from (S_{R_1})*, and the numbers in black are the monotonicity and normalization constraint of a capacity. Then we use the software VINCI to compute the volume of* (S_{C_1})*, which is 1.75e-02, which gives an acceptance rate* $r^a_n(S_C) = 0.371$*. Compared with the original acceptance rate* $r^a_n(S_R) = 0.097$*, it has been improved by a factor three.*

5.3 ECG with preference information

Now we would like to incorporate the system (S_C) to ECG, restricting to **constraint 1** with $\alpha = 0$. The principle of the ECG algorithm is to find the set of maximal elements at each step. If there exist element $S, S' \in 2^N$ with $S \not\subset S'$ and $\mu(S) \leq \mu(S')$, element S should be ranked before element S', hence element S cannot be selected in the set of maximal elements till element S' is removed from the poset.

For example, if we add a constraint like $\mu(\{1,3\}) \geq \mu(\{1,2\})$ to the algorithm, at each stage when we find the set of maximal elements, $\{1,2\}$ cannot be in the set until $\{1,3\}$ has been removed from the poset $(2^N, \subseteq)$ (shown in Figure 6 left). If we add a constraint $\mu(\{3\}) \geq \mu(\{1,2\})$ into the algorithm, at each stage, element $\{1,2\}$ cannot be in the set until all the elements that contain $\{3\}$ have been removed from the poset $(2^N, \subseteq)$ (shown in Figure 6 right).

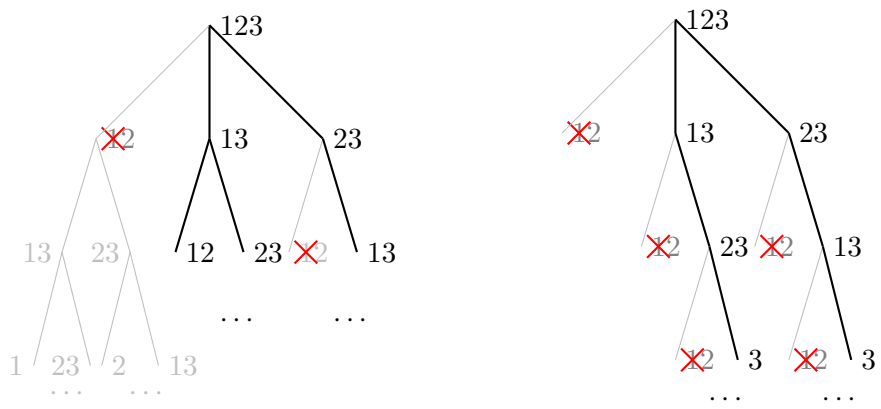

Figure 6: Left constraint: $\mu(\{1,3\}) \geq \mu(\{1,2\})$. Right constraint: $\mu(\{3\}) \geq \mu(\{1,2\})$ (which implies $\mu(\{1,3\}) \geq \mu(\{1,2\})$ and $\mu(\{2,3\}) \geq \mu(\{1,2\})$)

Revised algorithm

For the revised ECG algorithm, based on the original algorithm, we simply modify the maximal set ($\mathsf{Max}(P)$) and add two arrays describing the DM's preference information, called the *dominant* and the *dominated* arrays. For a given preferential information $(a,b) \in P$, we put a in the dominant array and b in the dominated array at the same position. For example, if we have $\mu(\{1\}) \leq \mu(\{2\})$ as the first restriction, then the first element of the dominated array is $\{1\}$, and the first element of the dominant array is $\{2\}$. Referring to Example 4 (system (S_{C_1})), we could only add $\mu(\{1\}) \leq \mu(\{2\})$ and $\mu(\{2\}) \leq \mu(\{1,3\})$ to ECG. Therefore, the dominant array would be $[\{2\},\{1,3\}]$ and the dominated array would be $[\{1\},\{2\}]$. We denote by S the set belonging to the *dominated* array, and by S' the corresponding set in the *dominant* array. This revised algorithm focuses on the fact that if $S \in \mathsf{Max}(P)$ and it exist $S'' \in \mathsf{Max}(P)$ with $S'' \supseteq S'$, then S should be eliminated from $\mathsf{Max}(P)$. Compared to the original algorithm, the modifications are highlighted in blue.

All-linear-extension$(P, AllLinearExtensions, count, dominated, dominant)$
Input: an array P containing a poset of size n, an array $AllLinearExtensions, count$, an array of dominated element and an array of dominant element
Output: All constraint linear extensions of poset P
If $|P| = 1$ **then**
    % When the bottom of a dendrogram is reached, add an empty linear extension to $AllLinearExtensions$.
    Append a zeros array of size n to $AllLinearExtensions$
    $AllLinearExtensions[count-1][n-1] \leftarrow P[0]$
    $count \leftarrow count + 1$
end if

% Remove the dominated elements from $\mathsf{Max}(P)$.
For i in range of length dominated **do**
    For S in $\mathsf{Max}(P)$ **do**
        If $S \supseteq dominant[i]$ and $dominated[i]$ in $\mathsf{Max}(P)$:
            Remove dominated[i] from $\mathsf{Max}(P)$
            break
For i in $\mathsf{Max}(P)$ **do**
    Remove i from P
    % recursion algorithm
    All-linear-extension$(P, AllLinearExtensions, count, dominated, dominant)$
    $AllLinearExtensions[count-1]$[size of P] $\leftarrow$ i
    Re-insert i to the end of poset P
end for

Experimental Results

We compare the execution time of the revised and original ECG with acceptance and rejection. For the experimental results, we add one by one the constraints listed below, i.e., $\mu(\{1\}) \geq \mu(\{2\})$ the first time, then $\mu(\{1,3\}) \geq \mu(\{4\})$ in addition the second time, etc.

$$\begin{cases} \mu(\{1\}) \geq \mu(\{2\}) \\ \mu(\{1,3\}) \geq \mu(\{4\}) \\ \mu(\{2,3\}) \geq \mu(\{3,4\}) \\ \mu(\{1,2,3\}) \geq \mu(\{2,4\}) \\ \mu(\{1,2,4\}) \geq \mu(\{2,3,4\}) \end{cases}$$

Table 7 gives the computation times required by the revised ECG algorithm. They are to be compared with the computation time for the original ECG with acceptance and rejection, which is always around 14s, not depending on the number of constraints.

number of restrictions	1	2	3	4	5
Time	6.8s	6.0s	3.0s	2.8s	1.4s

Table 7: Computational time for revised ECG when $n = 4$

The original ECG needs to generate all the linear extensions corresponding to poset $(2^N, \subseteq)$ and then filter them, whereas the revised algorithm only needs to generate the linear extensions that satisfy the constraints. Therefore, when we add more restrictions, the fewer the number of linear extensions, the less time the algorithm spends.

5.4 IRNG with preference information

We present in this section the revised algorithm of IRNG when incorporating the inequalities of the system (S_C), and illustrate it with Example 4. Throughout this section, we order the subsets of $2^N \setminus \{\varnothing, N\}$ by the cardinal-lexicographic order, that is, in increasing cardinality and using the lexicographic order for sets of same cardinality (e.g., for $n = 4$ we obtain 1, 2, 3, 4, 12, 13, 14, ..., 34, 123, 124, 134, 234, omitting braces and commas). We denote by $Ord(S)$ the rank of set S in this order (e.g., $Ord(13) = 6$ when $n = 4$). Hence we have always $Ord(\{1\}) = 1$ and $Ord(N \setminus \{1\}) = 2^n - 2$. We take the convention that in $v_{min}^{S,S'}$ and $v_{max}^{S,S'}$, always $Ord(S) < Ord(S')$.

Similar to ECG, the procedure of revised IRNG does not change, except that we need to reconsider $\mathsf{Max_p}\mu(S)$, $\mathsf{Min_p}\mu(S)$, $\mathsf{Max_p}\mathcal{R}\hbar(S)$ and $\mathsf{Min_p}\mathcal{R}\hbar(S)$ presented in Section 3.2. We introduce the following notation:

- Recall the value of already generated elements $S_1, \ldots, S_p$ are $\mu(S_1) = a_1, \ldots, \mu(S_p) = a_p$.
- We denote $\mathsf{Max_p}\mu(S)$, $\mathsf{Min_p}\mu(S)$ for revised IRNG by $\mathsf{Max}_\mathsf{p}^r\mu(S)$ $\mathsf{Min}_\mathsf{p}^r\mu(S)$.
- We denote the maximum and minimum values obtained from the inequalities of **constraint 1** in the system (S_C) as $\mathsf{Max}_\mathsf{p}^{c1}\mu(S)$ and $\mathsf{Min}_\mathsf{p}^{c1}\mu(S)$, where:

$$\mathsf{Min}_\mathsf{p}^{c1}\mu(S) = \max_{j\in\{1,\ldots,p\}}[(v_{min}^{S,S_j} + a_j)\mathbb{1}_{Ord(S)<Ord(S_j)}, (-v_{max}^{S_j,S} + a_j)\mathbb{1}_{Ord(S)>Ord(S_j)}].$$

$$\mathsf{Max}_\mathsf{p}^{c1}\mu(S) = \min_{j\in\{1,\ldots,p\}}[(v_{max}^{S,S_j} + a_j)\mathbb{1}_{Ord(S)<Ord(S_j)}, (-v_{min}^{S_j,S} + a_j)\mathbb{1}_{Ord(S)>Ord(S_j)}].$$

Then $\mathsf{Min}_\mathsf{p}^r\mu(S)$ and $\mathsf{Max}_\mathsf{p}^r\mu(S)$ becomes:

$$\mathsf{Min}_\mathsf{p}^r\mu(S) = \max(\mathsf{Min_p}\mu(S), v_{min}^S, \mathsf{Min}_\mathsf{p}^{c1}\mu(S)).$$

$$\mathsf{Max}_\mathsf{p}^r\mu(S) = \min(\mathsf{Max_p}\mu(S), v_{max}^S, \mathsf{Max}_\mathsf{p}^{c1}\mu(S)).$$

with v_{min}^{S,S_j}, v_{max}^{S,S_j}, v_{min}^S and v_{max}^S defined in section 5.2.2.

Example 4 continued: *We incorporate the preference information into the algorithm in the form of the following two-dimensional array denoted as $MinMax$. Then we use the system (S_{C_1}) to show the complete process of computing $\mathsf{Max}_\mathsf{p}^r\mu(S)$ and $\mathsf{Min}_\mathsf{p}^r\mu(S)$.*

Table 8: Minimum and Maximum value of system (3)

	$\mu(\{1\})$	$\mu(\{2\})$	$\mu(\{3\})$	$\mu(\{1,2\})$	$\mu(\{1,3\})$	$\mu(\{2,3\})$	$\mu(\{1\})-\mu(\{2\})$	$\mu(\{1\})-\mu(\{3\})$	...
Min	0.2	0	0	0.2	0.4	0	0	-0.25	...
Max	1	1	1	1	1	1	1	1	...

Let us suppose that we first randomly select element $\{2\}$ *from* $2^N \setminus \{N, \emptyset\}$ *as the first one, and assign a value* a_1 *to it following the IRNG algorithm with* $\mathsf{Max}^r\mu(\{2\}) = v_{max}^{\{2\}}$ *and* $\mathsf{Min}^r\mu(\{2\}) = v_{min}^{\{2\}}$. *Then we randomly select the second element* $\{1,3\}$, *we need to find the value of* $v_{min}^{\{1,3\}}$, $v_{max}^{\{1,3\}}$, $v_{min}^{\{2\},\{1,3\}}$ *and* $v_{max}^{\{2\},\{1,3\}}$ *from Table 8 which are 0.4, 1,* -1 *and 0, thus obtaining* $\mathsf{Min}_1^r\mu(\{1,3\}) = \max(0.4, a_1)$, *and* $\mathsf{Max}_1^r\mu(\{1,3\}) = \min(1, 1 + a_1) = 1$. *Then we assign a value* a_2 *to it following the IRNG algorithm and* a_2 *should be between* $\max(0.4, a_1)$ *and 1.*

node	min	max	value
$\{2\}$	0	1	a_1
$\{1,3\}$	$\max(0.4, a_1)$	1	a_2
$\dots$	$\dots$	$\dots$	$\dots$

As this process continues, the subsequent element shall always be compared with each of the preceding ones to find $\mathsf{Min}_\mathsf{p}^r\mu(S)$ *and* $\mathsf{Max}_\mathsf{p}^r\mu(S)$.

It is worth noting that if $v_{max}^{S,S'} \leq 0$ or $v_{min}^{S,S'} \geq 0$, we obtain that the element S should be ranked before element S' or after S', respectively. We formalize the order relation between $\mu(S)$ and $\mu(S')$ by the quantity

$$\mathcal{R}(S, S') = \begin{cases} v_{max}^{S,S'}, & \text{if } Ord(S) < Ord(S') \\ -v_{min}^{S',S}, & \text{if } Ord(S) > Ord(S'). \end{cases}$$

If $\mathcal{R}(S, S') \leq 0$, then $\mu(S) \leq \mu(S')$. In such cases, $\mathsf{Min}_\mathsf{p}\mathcal{R}\hbar(S)$ and $\mathsf{Max}_\mathsf{p}\mathcal{R}\hbar(S)$ should also be recalculated. Taking Example 1 presented in Section 3.2, consider now $S = \{2,3\}$ with $N = \{1,2,3,4,5\}$, and suppose we have $v_{max}^{\{1,3\},\{2,3\}} \leq 0$ from the system (S_C). Then the subset $\{1,3\}$ should be ranked before $\{2,3\}$. As we have $\mu(\{1,2\}) \leq \mu(\{1,3\})$ from the example, the subsets that are necessarily ranked before S are the following: $\{1\}, \{3\}, \{1,3\}, \{2\}, \{1,2\}$. Then we apply (6) to obtain the smallest possible ranking of $\{2,3\}$. To generalise $\overline{\mathcal{S}}_p(S)$, $\underline{\mathcal{S}}_p(S)$ with constraint (denoted as $\underline{\mathcal{S}}_p^c(S)$ and $\overline{\mathcal{S}}_p^c(S)$), we have

$$\underline{\mathcal{S}}_p^c(S) = \{S_j\,, j \in \{1, \dots, p\} \text{ s.t. } \exists i \in \{1, \dots, p\}, \{S_i \subseteq S \text{ or } \mathcal{R}(S_i, S) \leq 0\} \text{ and } a_j \leq a_i\} \cup \{S\}$$

is the set of already generated subsets that are necessarily ranked before S, and

$$\overline{\mathcal{S}}_p^c(S) = \{S_j\,, j \in \{1, \dots, p\} \text{ s.t. } \exists i \in \{1, \dots, p\}, \{S_i \supseteq S \text{ or } \mathcal{R}(S, S_i) \leq 0\} \text{ and } a_j \geq a_i\} \cup \{S\}$$

is the set of already generated subsets that are necessarily ranked after S.

Observe that the condition defining these collections are milder, therefore in general $\underline{\mathcal{S}}_p^c(S)$ and $\overline{\mathcal{S}}_p^c(S)$ are larger than the original ones without constraints.

In the end, we apply formulas (5) and (6) to obtain $\mathsf{Max}_\mathsf{p}\mathcal{R}\hbar(S)$ and $\mathsf{Min}_\mathsf{p}\mathcal{R}\hbar(S)$ with constraint denoted as $\mathsf{Max}_\mathsf{p}^r\mathcal{R}\hbar(S)$ and $\mathsf{Min}_\mathsf{p}^r\mathcal{R}\hbar(S)$.

Experimental result

Since it takes more time to compute $\mathsf{Min}^r_{\mathsf{p}}\mu([S])$ and $\mathsf{Max}^r_{\mathsf{p}}\mu([S])$ than in the original method, this revised algorithm could be slower, as shown in Table 9. Due to the difference between the polytopes induced by (S_R) and (S_C), we still need to apply the acceptance and rejection method after generating capacities by the revised algorithm. For Example 4 of Section 5.2.2, we generate 10,000 capacities compatible with the preference information (S_{R_1}), and the computation times are shown in the following table.

Table 9: Computational time for generating 10,000 capacities in **Example 4**

	revised IRNG	IRNG
before acceptance and rejection	1.6s	0.75s
after acceptance and rejection	6.7s	9.9s

From the table, we see that the revised IRNG algorithm takes about 2 times longer to generate a capacity than the original one. Therefore, if the ratio of the two acceptance rates is more than two, the revised IRNG is faster than the original method. Recall that the original acceptance rate of the system (S_{R_1}) is around 10% and the new acceptance rate of the system (S_{R_1}) from the system (S_{C_1}) is around 37%, hence the revised IRNG is faster than IRNG for Example 4.

For $n = 4$, let us use the same example as for the revised ECG's experimental result, by gradually adding the following restrictions:

$$\begin{cases} \mu(\{1\}) \geq \mu(\{2\}) \\ \mu(\{1,3\}) \geq \mu(\{4\}) \\ \mu(\{2,3\}) \geq \mu(\{3,4\}) \\ \mu(\{1,2,3\}) \geq \mu(\{2,4\}) \\ \mu(\{1,2,4\}) \geq \mu(\{2,3,4\}) \end{cases}$$

The computation times for revised IRNG and original IRNG are shown in the following table.

Table 10: Computational time for revised IRNG and IRNG when $n = 4$

number of restrictions	1	2	3	4	5
Revised IRNG	9.73s	11.66s	11.20s	12.62s	12.07s
IRNG	7.86s	9.33s	19.13s	21.47s	45.00s

From the Table 10, we notice that with the increase in the ratio of the two acceptance rates, revised IRNG is much faster than the original IRNG.

6 Concluding remarks

We have proposed an improved version of the random node generator of Havens and Pinar, by investigating in a deeper way the probability distribution of the coefficients $\mu(S)$. The results show that our algorithm yields distributions much closer to the exact ones, compared to the original random node generator, while keeping a very reasonable computation time, much smaller than the one required by the Markov Chain method.

In a second part, we have incorporated simple constraints on the capacity coefficients into our algorithms. This permits to take into account some preferential information given by the Decision Maker. We have shown that we can considerably improve the generation time compared to the naive approach of acceptance and rejection.

Further studies will be devoted to the generation of special families of capacities, like supermodular, k-additive capacities, etc.

7 Conflict of Interest

The authors declare that they have no conflict of interest

8 Data availability

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References

[1] S. Angilella, S. Corrente, and S. Greco. Stochastic Multiobjective Acceptability Analysis for the Choquet integral preference model and the scale construction problem. European Journal of Operational Research, 240:172–182, 2015.

[2] S. Angilella, S. Corrente, S. Greco, and R. Slowinski. Robust ordinal regression and stochastic multiobjective acceptability analysis in multiple criteria hierarchy process for the Choquet integral preference model. Omega, 63:154–169, 2016.

[3] C. A. Bana e Costa, J.M. De Corte, and J.-C. Vansnick. MACBETH. International Journal of Information Technology and Decision Making, 11:359–387, 2012.

[4] E. F. Combarro, I. Díaz, and P. Miranda. On random generation of fuzzy measures. Fuzzy Sets and Systems, 228:64–77, 2013.

[5] E. F. Combarro, J. Hurtado de Saracho, and I. Díaz. Minimals Plus: an improved algorithm for the random generation of linear extensions of partially ordered sets. Information Sciences, 501:50–67, 2019.

[6] G.Choquet. Theory of capacities. Annales de l'Institut Fourier,, (5):131–295, 1953.

[7] M. Grabisch. Set Functions, Games and Capacities in Decision Making, volume 46 of Theory and Decision Library C. Springer, 2016.

[8] M. Grabisch, I. Kojadinovic, and P. Meyer. Using the Kappalab R package for Choquet integral based multi-attribute utility theory. In Proc. Int. Conf. on Information Processing and Management of Unicertainty (IPMU'06), pages 1702–1709, Paris, France, July 2006.

[9] M. Grabisch and Ch. Labreuche. A decade of application of the Choquet and Sugeno integrals in multi-criteria decision aid. Annals of Operations Research, 175:247–286, 2010.

[10] M. Grabisch, Ch. Labreuche, and P. Sun. An approximation algorithm for random generation of capacities. To appear in Order, 2023. https://doi.org/10.1007/s11083-023-09630-0.

[11] S. Greco, V. Mousseau, and R. Słowinski. Ordinal regression revisited: Multiple criteria ranking with a set of additive value functions. European Journal of Operational Research, 191:416–436, 2008.

[12] T. C. Havens and A. J. Pinar. Generating random fuzzy (capacity) measures for datafusion simulations. In IEEE Symposium Series on Computational Intelligence (IEEE SSCI2017), pages 1–8, 2017.

[13] M. Herin, P. Perny, and N. Sokolovska. Learning sparse representations of preferences within Choquet expected utility theory. In 38th conference on Uncertainty in Artificial Intelligence (UAI'2022), Eindhoven, Netherlands, 2022.

[14] A. Karzanov and L. Khachiyan. On the conductance of order Markov chains. Order, 8:7–15, 1991.

[15] R. L. Keeney and H. Raiffa. Decision with Multiple Objectives. Wiley, New York, 1976.

[16] I. Kojadinovic, J.-L. Marichal, and M. Roubens. An axiomatic approach to the definition of the entropy of a discrete Choquet capacity. Information Sciences, 172:131–153, 2005.

[17] D.H. Krantz, R.D. Luce, P. Suppes, and A. Tversky. Foundations of measurement, volume 1: Additive and Polynomial Representations. Academic Press, 1971.

[18] R. Lahdelma, J. Hokkanen, and P. Salminen. SMAA – Stochastic Multiobjective Acceptability Analysis. European Journal of Operational Research, 106:137–143, 1998.

[19] K.De Loof, H.De Meyer, and B.De Baets. Exploiting the lattice of ideals representation of a poset. Fundamenta Informaticae, 71:309–321, 2006.

[20] T. Magoč and F. Modave. Optimization of the Choquet Integral Using Genetic Algorithm, pages 97–109. Springer International Publishing, Cham, 2014.

[21] B. Mayag, M. Grabisch, and Ch. Labreuche. A characterization of the 2-additive Choquet integral through cardinal information. Fuzzy Sets and Systems, 184:84–105, 2011.

[22] B. Mayag, M. Grabisch, and Ch. Labreuche. A representation of preferences by the Choquet integral with respect to a 2-additive capacity. Theory and Decision, 71:297–324, 2011.

[23] R.Stanley. Two poset polytopes. Discrete and Computational Geometry, (1):9–23, 1986.

A The Markov Chain method

This method is due to Karzanov and Khachiyan [14]. Consider a given poset P, and E the set of linear extensions of P. Two linear extensions $X_1, X_2 \in E$ are said to be neighbors, if X_2 can be obtained by a single transposition of two consecutive elements in X_1, that is $X_1 = (x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_{|P|})$ and $X_2 = (x_1, x_2, \ldots, x_{i+1}, x_i, \ldots, x_{|P|})$ for $i \in [1, |P|-1]$. Consider a Markov chain with E being the set of states. The transition probability between two states X_1 and X_2 is given by:

$$p(X_1, X_2) = \begin{cases} 1/(2|P|-2) & \text{If } X_1 \text{ and } X_2 \text{ are neighbors} \\ 1 - n(X_1)/(2|P|-2) & \text{If } X_1{=}X_2 \\ 0 & \text{otherwise} \end{cases}$$

where $n(X_1)$ denotes the number of neighbors of X_1. The Markov chain with the above transition matrix describes a random walk through the simplices, and this random walk starts at an arbitrary simplex X_0 in the triangulation, then there is a move to a new simplex with probability $\frac{1}{2n-2}$. Karzanov and Khachiyan proved that the transition matrix induces an ergodic time-reversible Markov chain with uniform stationary distribution, that means for an arbitrary initial probability distribution on E, after T steps (T big enough), the distribution converges to the uniform distribution on E. Thus, for a given poset P and sufficiently large T, the following algorithm gives a nearly uniform generator of linear extensions of the poset.

Algorithm Classical Karzanov-Khachiyan chain

1. Input X_0 as the initial state (an arbitrary linear extension).
2. Randomly select $r \in \{0, 1\}$.
3. If $r = 1$, randomly select i from $\{1, 2, 3, \ldots, |P| - 1\}$.
4. Interchange the value of i^{th} position and $(i + 1)^{\text{th}}$ position of X_0, if the new X_0 belongs to the set of linear extensions E.
5. Repeat **T** times Step 2,3,4, and output new X_0.