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[1,2], and Christophe LABREUCHE[3]

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

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 [4, 5]. It is often useful in practice to be able to randomly generate capacities, in a uniform way (measure of performance of models, learning/identification phase, etc.). This problem reveals to be surprisingly difficult, because 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 $(2^N, \subseteq)$ [9]. However, the number of linear extensions of $(2^N, \subseteq)$ grows

⋆ Corresponding author

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 [8], Combarro et al [1, 2], and also the authors of this paper [6]. 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 [7]. 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 aim of this paper is to provide an improvement of the random node generator of Havens and Pinar, 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.

The paper is organized as follows: Section 2.1 explains the basic facts on the random generation of capacities and describes the random node generator as well as the exact method based on linear extensions. In Section 2.2, we investigate the theoretical distribution of the coefficients of a capacity, and in Section 2.3 we describe our improved random node generator. Section 3 gives experimental results on the comparison of various methods. Section 4 concludes the paper.

2 Random Node generator based on Beta distribution

2.1 Background

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* [9] 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 [9] 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. Let $N := \{1, 2, \ldots, n\}$ be a finite set of n elements. A *(normalized) capacity* [3,4,10] on N is a set function $\mu : 2^N \to [0, 1]$ satisfying $\mu(\varnothing) = 0, \mu(N) = 1$ (normalization), and the property $S \subseteq T \Rightarrow$

$\mu(S) \leq \mu(T)$ (monotonicity). It is easy to see that the set of capacities, denoted by $\mathcal{C}(N)$, 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 $n = 8$. 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.

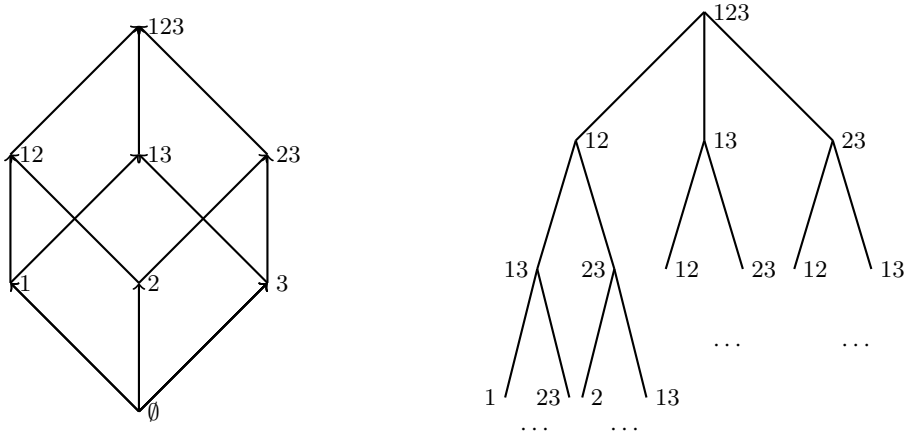

Fig. 1. Case $n = 3$. Left: representation of the poset $(2^N, \subseteq)$. Right: Dendrogram of the maximal elements when running the procedure for generating all linear extensions using the DFS algorithm.

Algorithm 1

Exact-capacity-generator(n, k)

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 like the Markov Chain method [8], the 2-level approximation method [6], etc., or based on other principles like the *Random Node generator (RNG) algorithm* introduced by T. C. Havens and A. J. Pinar in [7]. 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.

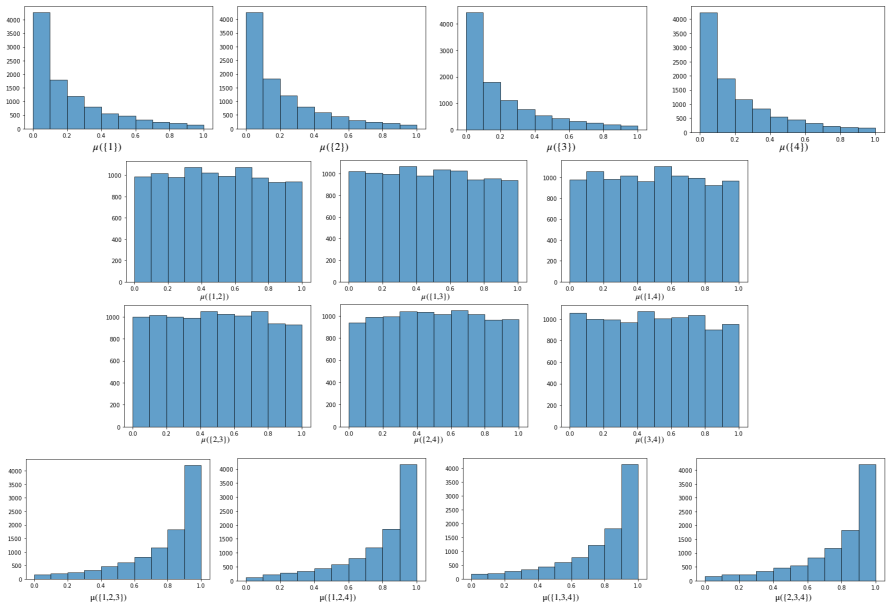

Fig. 2. Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by RNG (compare with Fig. 3 where the exact generator has been used).

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.

2.2 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.

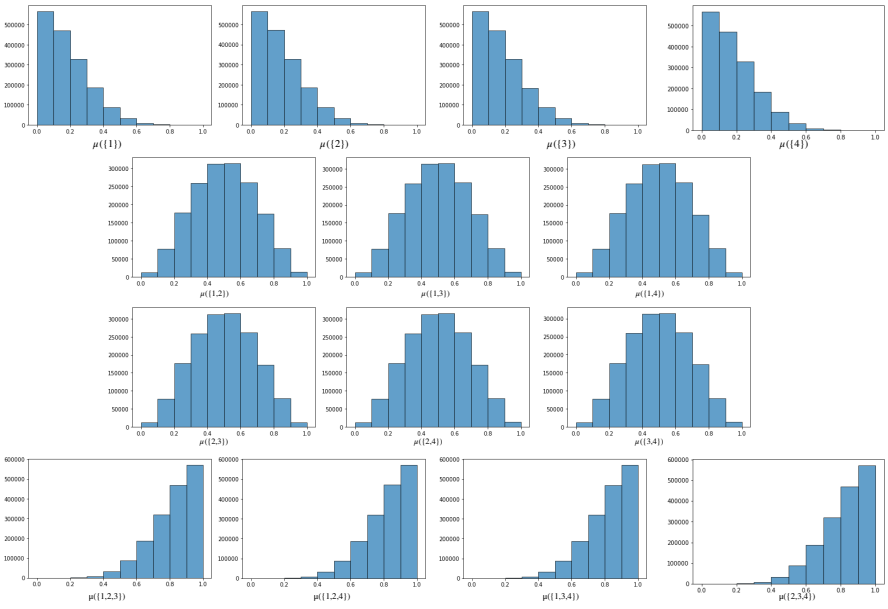

Fig. 3. Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by ECG.

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_is 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., $\mathcal{R}\hbar(N) = 2^n - 1$.

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 $\mu(S)$

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

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

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

with $F_{\mu_{(i)}}(x)$ the cumulative distribution function of $\text{Beta}(i, 2^n - 1 - i)$, $\min(\mathcal{R}\hbar(S))$ the smallest possible ranking of $\mu(S)$ and $\max(\mathcal{R}\hbar(S))$ 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

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

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

$$f_{\mu(S)}(x) = \sum_{i=min(\mathcal{R}\hbar(S))}^{max(\mathcal{R}\hbar(S))} f_{\mu_{(i)}}(x) \times \mathbb{P}(\mathcal{R}\hbar(S) = i) \tag{1}$$

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

Density (1) 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 (1), 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 $[\mathsf{Min_p}\mu(S), \mathsf{Max_p}\mu(S)]$ where

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

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,

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

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

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

is the set of already generated subsets that are necessarily ranked after S (including S). The smallest possible ranking $\mathsf{Min_p}\mathcal{RK}(S)$ 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 $\underline{\mathcal{S}}_p(S) := \{\underline{S}_1, \ldots, \underline{S}_q\}$. 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

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

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 (2), 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

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

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

$$\begin{aligned}&\mathsf{Max_p}\mathcal{R}\hbar(S) = 2^n-1-\left|\{T\supseteq\overline{S}_j,\ T\neq N \text{ and } j\in\{1,\dots,q'\}\right|\\&=2^n-1-\sum_{k=1}^{q'}\left((-1)^{k-1}\sum_{1\le i_1<i_2<\cdots<i_k\le q'}\left(2^{|N\setminus(\overline{S}_{i_1}\cup\cdots\cup\overline{S}_{i_k})|}-1\right)\right).\end{aligned}\tag{3}$$

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 $\overline{\mathcal{S}}_p(\{1,2,5\}) = \big\{\{1,2,3\},\{1,3,4\},\{1,2,4,5\},\{1,2,5\}\big\}$. 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 (3), 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:

$$\begin{aligned}&\mathbb{P}(\mu(S)\le x|\mu(S_1)=a_1,\dots,\mu(S_p)=a_p)\\&=\sum_{i=min_p\mathcal{R}\hbar(S)}^{max_p\mathcal{R}\hbar(S)}\mathbb{P}(\mathcal{R}\hbar(S)=i|\mu(S_1)=a_1,\dots,\mu(S_p)=a_p)\\&\qquad\times\mathbb{P}(\mu(S)=\mu_{(i)}\le x|\mu(S_1)=a_1,\dots,\mu(S_p)=a_p)\end{aligned}\tag{4}$$

with

$$\begin{aligned}&\mathbb{P}(\mathcal{R}\hbar(S)=i|\mu(S_1)=a_1,\ldots,\mu(S_p)=a_p)\\&\approx\mathbb{P}(\mathcal{R}\hbar(S)=i|\mathsf{Min_p}\mathcal{R}\hbar(S)\leq\mathcal{R}\hbar(S)\leq\mathsf{Max_p}\mathcal{R}\hbar(S))\end{aligned}\tag{5}$$

and

$$\begin{aligned}&\mathbb{P}(\mu(S)=\mu_{(i)}\leq x|\mu(S_1)=a_1,\ldots,\mu(S_p)=a_p)\\&=\mathbb{P}(\mu_{(i)}\leq x|\mathsf{Min_p}\mu(S)\leq\mu_{(i)}\leq\mathsf{Max_p}\mu(S))\end{aligned}\tag{6}$$

2.3 The improved random node generator

Thanks to the previous considerations and Equations (4), (5) and (6), 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 (4), computed through (5) and (6).

According to Equation (6), 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 (5), 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 propability is stored in array *probability* (where $probability[S][i]=\mathbb{P}(\mathcal{R}\hbar(S)=i)$) in the following algorithm. 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 [8], 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. Once we have obtained these probabilities, we store them in a file so that they can be used repeatedly.

Algorithm 3

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

Input: a poset P of $2^N\setminus\{\varnothing,N\}$, 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 ←[ ],[ ]
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: $\mathscr{L} \leftarrow$ an array of elements of $2^N \setminus \{\varnothing, N\}$ in random order
4: $p \leftarrow 0$
5: **for** S in $\mathscr{L}$ **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: $\Pr_{min} \leftarrow \sum_{j=0}^{\mathsf{Min_p}\mathcal{R}\hbar(\mu([s]))-1} probability[s][j]$
9: $\Pr_{max} \leftarrow \sum_{j=\mathsf{Max_p}\mathcal{R}\hbar(\mu([s]))+1}^{2^n-3} probability[s][j]$
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: $r \leftarrow \Pr_{min} + (1 - \Pr_{max} - \Pr_{min}) * r$
13: $Rank \leftarrow \mathsf{Min_p}\mathcal{R}\hbar(S))$
14: $\Pr \leftarrow \Pr_{min}$
15: **While** $r > \Pr$ **do**
16: $\Pr \leftarrow \Pr + probability[S][Rank]$
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 ot IRNG. The $2^n - 2$ subsets are ordered in array $\mathscr{L}$. 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 (2) and (3));
- $l.8 - 9$: the computation of $\Pr_{min}$ and $\Pr_{max}$ requires at most 2^n operations;
- We assume that the While loop in $l.10$ is run at most M times. The While loop in $l.15$ is run at most 2^n times. Then the complexity of $l.10 - 18$ is $M \times 2^n$.

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.

3 Experimental results

We compare the performance of the IRNG with the RNG and Markov Chain generator. We apply the Markov Chain method to obtain $\mathbb{P}(\mathcal{R}k(S) = i)$ for all the following experiments. In the experiments, we limit ourselves to $n = 4$ in order to be able to compare the results with the ECG. Figure 4 shows the distribution of $\mu(S)$ generated by the IRNG for $n = 4$.

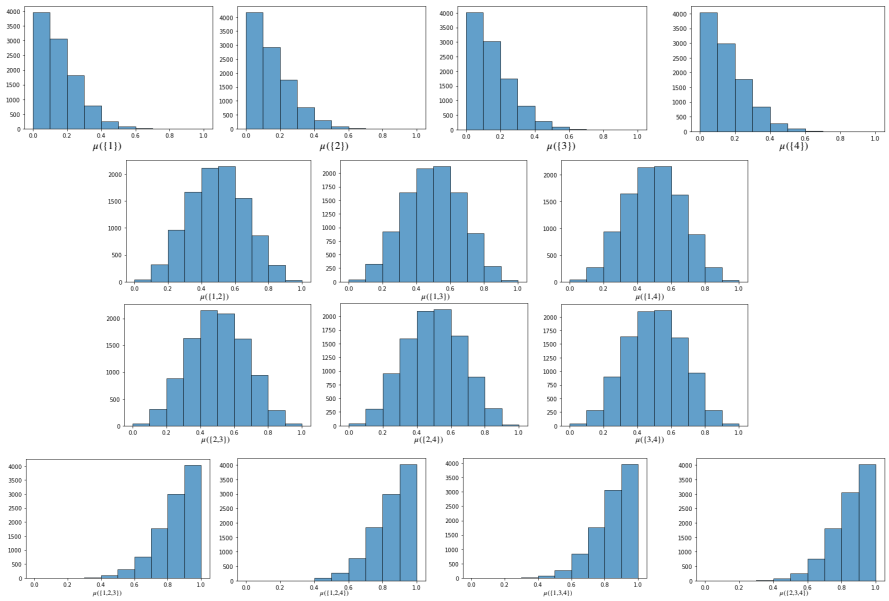

Fig. 4. Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by IRNG (compare with Fig. 3 where the exact generator has been used).

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:

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

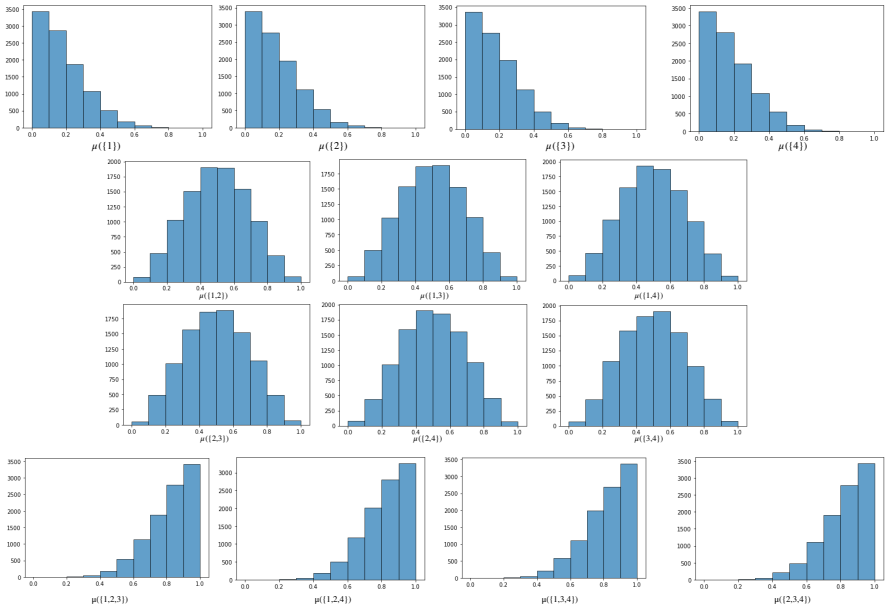

Fig. 5. Case $n = 4$. Histograms of the values of $\mu(S)$, $S \in 2^N \setminus \{N, \varnothing\}$, generated by the Markov chain generator (compare with Fig. 3).

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 (5) is not taken into account, as they are computed once for all off line.

From Table 1, we compute the sum of the Kullback-Leibler divergences 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 0.384 and for Markov Chain is 0.132. As can be seen from these results, compared to the RNG, the distribution of μ obtained from

capacity generator	$\mu(\{1\})$	$\mu(\{2\})$	$\mu(\{3\})$	$\mu(\{4\})$
RNG	0.4220	0.3651	0.3708	0.3947
IRNG	0.0376	0.0392	0.0356	0.0367
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	0.0223	0.0187	0.0222	0.0253	0.0178	0.0191
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	0.0296	0.0270	0.0270	0.0258
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

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	25.270	243

Table 2. Comparison of CPU time for generating 10000 capacities

Unlike RNG, IRNG needs to compute $\mathsf{Min_p}\mathcal{R}\!\!\!/(S)$ and $\mathsf{Max_p}\mathcal{R}\!\!\!/(S)$ 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, and not much different from that of the Markov Chain Generator.

4 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.

Further studies will be devoted to the generation of special families of capacities, as well as generation under additional constraints on the capacities.

References

1. Combarro, E.F., Díaz, I., Miranda, P.: On random generation of fuzzy measures. Fuzzy Sets and Systems **228**, 64–77 (2013)
2. Combarro, E.F., Hurtado de Saracho, J., Díaz, I.: Minimals Plus: an improved algorithm for the random generation of linear extensions of partially ordered sets. Information Sciences **501**, 50–67 (2019)
3. G.Choquet: Theory of capacities. Annales de l'Institut Fourier, (5), 131–295 (1953)
4. Grabisch, M.: Set Functions, Games and Capacities in Decision Making, Theory and Decision Library C, vol. 46. Springer (2016)
5. Grabisch, M., Labreuche, C.: A decade of application of the Choquet and Sugeno integrals in multi-criteria decision aid. Annals of Operations Research **175**, 247–286 (2010)
6. Grabisch, M., Labreuche, C., Sun, P.: An approximation algorithm for random generation of capacities. arXiv:2206.04774 (2022)
7. Havens, T.C., Pinar, A.J.: Generating random fuzzy (capacity) measures for data-fusion simulations. In IEEE Symposium Series on Computational Intelligence (IEEE SSCI2017) pp. 1–8 (2017)
8. Karzanov, A., Khachiyan, L.: On the conductance of order Markov chains. Order **8**, 7–15 (1991)
9. R.Stanley: Two poset polytopes. Discrete and Computational Geometry (1), 9–23 (1986)
10. Sugeno, M.: Theory of fuzzy integrals and its applications. Ph.D. thesis, Tokyo Institute of Technology (1974)