ATLANTIS PRESS
*Atlantis Studies in Uncertainty Modelling, volume 3*
**Joint Proceedings of the 19th World Congress of the International Fuzzy Systems Association (IFSA), the 12th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT), and the 11th International Summer School on Aggregation Operators (AGOP)**

# Empirical Monotonicity of Non-deterministic Computable Aggregations

**Luis Garmendia**[a] and **Daniel Gómez**[b] and **Luis Magdalena**[c] and **Javier Montero**[d]
[a] Facultad de Informática, Universidad Complutense de Madrid, lgarmend@fdi.ucm.es
[b]Facultad de Estudios Estadísticos, Universidad Complutense de Madrid, dagomez@estad.ucm.es
[c]ETSI Informáticos, Universidad Politécnica de Madrid, Spain, luis.magdalena@upm.es
[d]Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid, monty@mat.ucm.es

## Abstract

The concept of aggregation has been usually associated with that of aggregation functions, assuming that any aggregation process can be represented by a function. Recently, computable aggregations have been introduced considering that the core of the aggregation processes is the program that enables it. In this new framework, the concept of monotonicity of an aggregation, linked to the monotonicity of the function defining the aggregation, should be revisited once there is not such a function. The new concept of aggregation also opens a new scenario where the aggregation can even be non-deterministic.

Assuming these premises, the present work focuses on monotonicity of non-deterministic computable aggregations, considering the situation where the program implementing the aggregation is a black box, that is, only inputs and outputs are available. But due to non-determinism, a certain input will not always produce the same output, and the analysis should be based on multiple executions of the program, generating for that single input a collection of outputs (a list) that could be analysed as such, or interpreted as a distribution.

Monotonicity analysis will require the comparison of those lists of outputs in terms of ordering, and to do so it is needed the previous definition of an order relation in the set of lists.

**Keywords:** Aggregation, Computable aggregation, non-deterministic aggregation, partial orders of lists, monotonicity.

## 1 Introduction

Aggregation is a fundamental part of decision, compression and summarization processes considering complex information [2, 3, 4, 7].

Usually, aggregation processes have been associated in literature with aggregation functions. An aggregation process was represented by a function (or a family of functions). Recently, the association between aggregation processes and functions was broken in [12] with the definition of computable aggregations. In that paper, the authors put the emphasis in the programs enabling the aggregation processes, not necessarily being expressed in terms of functions. Given a function that describes an aggregation process, it can be implemented in many ways resulting in significant differences. This approach focuses on the way each aggregation is obtained, i.e., the procedure that produces the aggregated value. Relevant properties come from these procedures, and it is the specific procedure we apply what should be the main object of study.

It is even possible that the program (the considered procedure) does not implement any function. The rupture between functions and aggregation operators allows to open the domain of aggregation processes to a field not yet analysed in this discipline: non-deterministic computable aggregations [6]. Those aggregation processes where we cannot guarantee that the obtained result will coincide if we replicate the process, that is, considering the same input information we can obtain a different output. Such an aggregation process can not be described in terms of a function (a mapping). This type of aggregation procedure is very common in statistics, where, due to the volume of information that is processed, it is frequent to choose a representative sample on which it is operated. Obviously, replicating the process does not imply that the sample coincides and therefore the result varies. This kind of computable aggregations need to redefine some common properties of the aggregation operator func-

tions that are no longer valid in non-deterministic computable aggregations.

The purpose of this paper is to revisit and adapt the concept of monotony (as a central property of aggregation processes) for non-deterministic computable aggregations. Considering the probabilistic nature of some of the methods applied in the paper, it would be possible to approach the concept from an stochastic point of view, in terms of distributions (as in [10]), even assuming some additional knowledge on the aggregation process and analysing the theoretical distributions of the generated outputs. But in the present case the redefinition will only consider input-output information, without assuming a deeper knowledge of the program itself. That will allow us to consider even those cases where the aggregation process (the program) is a black box. To do so, the analysis of monotonicity will be approached from an empirical point of view.

The paper is structured as follows. Section 2 contains some background and preliminary knowledge on computable aggregations and non-deterministic computable aggregations. Section 3 considers different options for characterising a computable aggregation (either deterministic or not) in terms of its outputs considered as a list, the distribution describing that list, or even the theoretical distribution corresponding to that output. Section 4 defines and compares some orders on the set of non-empty finite lists in $[0,1]$, that will be considered to compare the outputs of a computable aggregation. Section 5 extends the concept of monotonicity for non-deterministic computable aggregations, considering the case where those computable aggregations are described in terms of lists of outputs. Finally, Section 6 presents some conclusions and future works.

## 2 Preliminaries

### 2.1 Computable aggregations

Although it is not the only option, aggregation operators [1, 5, 8, 13, 14] are usually associated to the use of membership functions, and this is the reason why they are usually defined as follows:

**Definition 1.** [2] An aggregation operator is a mapping $Ag : [0,1]^n \to [0,1]$ that satisfies:

1. $Ag(0,0...,0) = 0$ and $Ag(1,1,..,1) = 1$.
2. $Ag$ is monotonic.

Let us observe that this definition presents an aggregation operator as a function that is linked to the value of $n$. There is a different function for each $n$. Other authors (see [2] for example) present an aggregation operator as a function that considers any cardinality for the set of items to be aggregated, defining $Ag$ as a function $Ag : \cup_n [0,1]^n \to [0,1]$.

In [12], it was introduced the concept of computable aggregation focusing on the idea that in aggregation processes we should pay our attention in the program that makes possible the aggregation, better than in a generic function that has not yet been implemented.

In order to formally introduce computable aggregations, we previously need to introduce the concepts of program, list and algorithm.

**Definition 2.** A *list L* is an abstract data type that represents a sequence of values. A list can be defined by its behaviour, and its implementation must provide at least the following operations: test whether a list is empty, add a value, remove a value, and compute the length of a list (number of elements).

**Definition 3.** [11] In mathematics and computer science, an *algorithm* is a self-contained step-by-step set of operations to be performed.

Algorithms can be expressed using quite different notations, including natural languages, pseudo-code, flowcharts, drakon-charts, programming languages, and control tables.

**Definition 4.** [15] A computer program, or simply a *program*, is a sequence of instructions written to perform a specific task on a computer.

Considering that this paper will use a list notation, Definition 1 can be reinterpreted as follows.

**Definition 5.** Let $L$ be the set of non-empty and finite lists of degrees in $[0,1]$. Then an *aggregation operator* is a mapping $Ag : L \to [0,1]$ that satisfies:

1. $Ag(0,0...,0) = 0$ and $Ag(1,1,..,1) = 1$.
2. $Ag$ is monotonic.

It is even possible in some cases to define aggregation processes going beyond functions by considering methods that do not match with the concept of function.

To analyse this option let us remind the concept of computable aggregation ([12]) focusing on the idea that in aggregation processes we should pay our attention in the program that makes possible the aggregation instead of a generic function that has not yet been implemented. The main contribution in [12] was to separate the strong association that existed between "aggregation process" and explicit functions.

**Definition 6.** [6] Let $L < T >$ be a finite and non-empty list of n elements with type $T$. A *computable aggregation* is a program $P$ that transform the list $L < T >$ into an element of $T$.

In [9], it was proofed that any classical aggregation process (aggregation operators, pre-aggregations, fusion functions or extended fusion functions) can be implemented by an algorithm and by a program.

However, given a program, there exist some situations in which it is not possible to build a function associated to it. This is the case, for example, of non-deterministic programs.

It is important to emphasize that the computable aggregation paradigm broadens the set of possible aggregation methods, being in this case not limited to those methods for which there is a function that explains the aggregation process.

**Example 1.** Let's consider a population $X$ of size $n$ ($X = \{x_1, \ldots, x_n\}$). Suppose that the objective of the aggregation process is to estimate the average value of $X$. If $n$ is too large considering the available time for computing the aggregation, a reasonable solution would be to estimate the average value through statistical sampling. In such a situation, $k$ elements of the population will be drawn at random to further calculate the average, i e., given the set $\{x_1, ..x_n\}$, we choose $\{x_{i1}, \ldots, x_{ik}\}$ at random, computing the arithmetic mean of these $k$ elements. Obviously, this aggregation process can't be defined by means of an explicit function since the same input does not always produce the same output. This is a computable aggregation that can't be modelled as an aggregation function, and it is important to integrate such a situation in aggregation processes.

From previous example we can distinguish between those computable aggregations in which there exist an explicit function, from those in which this explicit function does not exist.

To go one step forward in our analysis, let's consider now the idea of deterministic and non-deterministic programs.

**Definition 7.** A program $P$ is a *deterministic program*, also known as repeatable, if it produces the very same output when given the same input, no matter how many times it is run.

Obviously, a program or an algorithm are non-deterministic when they do not match with the previous definition.

It is clear according to the definitions that a non-deterministic algorithm will always produce a non-deterministic program. Consequently, when analysing if a computable aggregation is deterministic or non-deterministic we should simply consider the corresponding Program.

**Definition 8.** A computable aggregation $P$ over the set $T$ is a *non-deterministic computable aggregation* if and only if the program implementing it is non deterministic.

From this definition we will say that a computable aggregation $P$ is deterministic when the program implementing it is deterministic, implying that the underlying algorithm is also deterministic.

### 2.2 Some non deterministic computable aggregations

Our interest relies on programs describing aggregation processes that are intrinsically non-deterministic, processes where, as an example, random or probabilistic decisions are involved. Some of them are the following [6]:

**Definition 9** (Probability sampled computable aggregation)**.** Given a value $p \in (0,1]$, and given a family of aggregation operators $\{Ag_n : [0,1]^n \longrightarrow [0,1],\ n \geq 1\}$, let us define the computable aggregation $P_{Ag,p}$ as the three steps program that for a given list $L_1 = (x_1, \ldots, x_n) \in L < [0,1] >$ performs the following actions:

- **Step 1**. To reduce the list $L_1$ into another list $L_2$ of lower (or equal) dimension by randomly removing each element (of the list) with probability $1 - p$.
- **Step 2**. If list $L_2$ is empty repeat Step 1.
- **Step 3**. Return the value $Ag_{|L_2|}(L_2)$, where $|L_2| \geq 1$.

Note that the computable aggregation $P_{Ag,p=1}$ becomes the program associated with the family of aggregation operators $\{Ag_n,\ n \geq 1\}$ and is deterministic since for $p = 1$, results $L_2 = L_1$.

Another way to build a class of non deterministic computable aggregations would be to fix a value $k$, and randomly select $k$ elements from the list, as those to be aggregated. Given a list of $n$ elements of $[0,1]$, $L_1 = \{x_1, \ldots, x_m\} \in < [0,1] >$, let us denote by $Sel_k$ a program that, if $m > k$, randomly chooses a sample (without re-sampling) of $k$ elements of the list, and maintains the same list when $m \leq k$.

**Definition 10** ($k$-sampled computable aggregation)**.** Given a family of aggregation operators $\{Ag_n\ n \geq 1\}$, the computable aggregation $P_{Ag,k}$, with $0 < k \leq n$, is

defined as the program that for a given list $L_1$ of $m$ elements, first applies the procedure $L_2 = Sel_k(L_1)$, and then computes the value $Ag_{|L_2|}(L_2)$.

Note that in the previous definitions the sampled list of degrees $L_2$ is contained in $L_1$.

**Definition 11** (Noisy computable aggregation)**.** Given a normal distribution $N(\mu,\sigma)$ and a family of aggregation operators $\{Ag_n : [0,1]^n \longrightarrow [0,1],\ n > 1\}$, we define the **noisy computable aggregation** $P_{N(\mu,\sigma),Ag}$ as the program that for any list $L_1 \in < [0,1] >$, with $|L_1| \leq n$, returns $P_{N(\mu,\sigma),Ag} = Ag(L_2)$, where $L_2$ is the list generated by replacing each element in $L_1$ with the same element after being modified by adding noise generated by the normal distribution (truncated to 0 or 1 in case that the resulting value was out of the $[0,1]$ interval).

The previous definitions just explore some options to design non-deterministic computable aggregations, but obviously are not the only ones.

## 3 Characterizing non deterministic computable aggregations

In previous papers we have analysed some structural properties of a computable aggregation (as recursivity [9]) by relating them to properties of the program implementing the aggregation. Some other properties could be analysed in terms of input-output relations. As an example, monotonicity relies on this kind of relation, and consequently could be somehow explored by considering the input-output relations generated by the program. The problem when we enter into the area of non-deterministic computable aggregations, is that the input-output relation is also non-deterministic, so any analysis requires some previous considerations.

Given a computable aggregation $P$ that aggregates a list $L_1 \in L < T >$, how can we characterise the output? A first option could be running the program several times and compiling the outputs. In case our computable aggregation being deterministic, all outputs should be the same. However, a non-deterministic computable aggregation would probably produce several different values.

From now on we will denote by $LP^m_{L_1} \in L < T >$ the list of length $m$ obtained after $m$ executions of the computable aggregation (program) $P$, over the list $L_1$. These $m$-realizations could be characterized in terms of a distribution.

**Definition 12** (Empirical distribution of a computable aggregation)**.** Given a computable aggregation $P$, and given a list $L_1 \in L < T >$, the distribution of results obtained after $m$ executions of $P$ over $L_1$, will be referred as the *empirical distribution with size m of the computable aggregation P, over the list* $L_1$ *in* $L$, represented by $DP^m_{L_1}$.

It is also possible that by simply analysing the program we can determine what should theoretically be the distribution of outputs for a given input.

**Definition 13** (Theoretical distribution of a computable aggregation)**.** Given a computable aggregation $P$ that aggregates a list $L_1 \in L < T >$, let us denote by $\mathscr{P}(L_1)$ the *theoretical distribution* after all possible realizations of $P$ over the fixed list $L_1$.

Obviously, if the computable aggregation $P$ is deterministic, the associated distribution $\mathscr{P}(L_1)$ will be a single value for each input list. For non deterministic programs, we will have here a probability distribution $\mathscr{P}(L_1)$ for each input $L_1$.

In general, given a non deterministic computable aggregation $P$, it is not possible to know the theoretical distribution $\mathscr{P}(L_1)$. Nevertheless, we could try to approximate it by considering $DP^m_{L_1}$.

In order to show the differences between empirical, theoretical and list of execution ($m$-realizations), we present the following example.

**Example 2.** Let $L_1 =< 0.1, 0.3, 0.5, 0.7 >$ be a list of four elements, and let $P$ be a deterministic program that calculates the average of the elements of the list. Let $m = 5$ be the number of executions. In one hand, it is easy to see that $LP^5_{L_1} =< 0.4, 0.4, 0.4, 0.4, 0.4 >$ is a list of 5 elements each one of them takes the value 0.4, since the program is deterministic and always gives the same result. On the other hand, the set $DP^5_{L_1} = \mathscr{P}(L_1) = \{0.4\}$

To analyse the aggregation process implemented by a non deterministic computable aggregation, we have several components to consider. There is a list $L_1 \in L < T >$ of elements to be aggregated. There is also a family of aggregation operators ($\{Ag_n\}$) underlying in the non deterministic aggregation process. Finally, two distributions describe the aggregation process: the theoretical distribution $\mathscr{P}(L_1)$ and the empirical distribution ($DP^m_{L_1}$). It is obvious that the interactions and relations among these elements will describe and characterize a non deterministic computable aggregation.

In this paper we will explore the monotonicity condition, consequently, if we assume that the output for a certain input list $L_1 \in L < T >$ is described in terms of $m$-realizations $LP^m_{L_1} \in L < T >$, being also a list, any monotonicity analysis will require a method to compare/order elements of $L < T >$.

## 4 Orders on lists of degrees of the same length

Let $L$ be the set of non-empty and finite lists of degrees in $[0,1]$ that could also be represented as $L < [0,1] >$. Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ be two lists of $L$ with the same length $n > 0$. We will define some relations in $L$ allowing the comparison of the lists $L_1$ and $L_2$. For this purpose we will use orders and preorders.

A preorder is a binary relation being reflexive and transitive, while orders are, in addition, antisymmetric. In fact, both, equivalence relations and order relations are special cases of preorders: an antisymmetric preorder is an order, while a symmetric preorder is an equivalence relation.

### 4.1 Max-Min preorder

Two lists of degrees in $L$ are Max-Min comparable if the maximum degree of one of them is lower than (or equal to) the minimum degree of the other.

**Definition 14** (Max-Min preorder)**.** A list $L_1 = [a_1,\ldots,a_n]$ in $L$ is Max-Min lower or equal than a list $L_2 = [b_1,\ldots,b_n]$ in $L$, if and only if the highest degree in $L_1$ is lower or equal than the minimum value of $L_2$. It is:

$$L_1 \leq_{\text{MaxMin}} L_2 \text{ if and only if } a_i \leq b_j, \forall i,j = 1,\ldots n.$$

This definition presents a relation that is not antisymmetric. Two lists of different length but with all their elements taking the same value are related in both directions (with $L_1 = [0.5,0.5]$ and $L_2 = [0.5,0.5,0.5]$, we have $L_1 \leq_{\text{MaxMin}} L_2$ and $L_2 \leq_{\text{MaxMin}} L_1$, but $L_1 \neq L_2$). Consequently the relation does not define an order but a preorder.

### 4.2 1to1 partial order

The previous condition is quite restrictive, so it is possible to relax it and compare the elements of the two lists just one by one. For this purpose we will first sort both lists in increasing order.

Two lists of degrees in $L$ are *sorted one to one* comparable if, for all integer $k$ from one to the length of the lists, the $k^{\text{th}}$ degree of one of the sorted lists (represented by $s(L_1)$) is always lower or equal to the corresponding $k^{\text{th}}$ degree of the other sorted list (represented by $s(L_2)$).

**Definition 15** (Sorted 1to1 preorder)**.** A list $L_1$ in $L$ is S1to1 lower or equal than a list $L_2$ in $L$, if and only if the $k^{\text{th}}$ degree of $s(L_1)$ is lower or equal to the $k^{\text{th}}$ degree of $s(L_2)$, for all $k$ from 1 to $n$, that is:

$$L_1 \leq_{\text{S1to1}} L_2 \text{ if and only if } s(L_1)_k \leq s(L_2)_k \text{ for all } k.$$

It would also be possible to compare the lists without previously sorting then. In that case the obtained **1to1 partial order** would be equivalent to ordering the lists as vectors in $\mathbb{R}^{|L_1|}$, that is,

$$L_1 \leq_{\text{1to1}} L_2 \text{ if and only if } L_{1k} \leq L_{2k} \text{ for all } k.$$

### 4.3 Aggregation preorder

**Definition 16** (Aggregation preorder)**.** Let $Ag : L \to [0,1]$ be an aggregation operator. A list $L_1 = [a_1,\ldots,a_n]$ in $L$ is $Ag$-lower or equal than a list $L_2 = [b_1,\ldots,b_n]$ in $L$, if and only if the aggregation of $L_1$ is lower or equal than the aggregation of $L_2$. That is:

$$L_1 \leq_{Ag} L_2 \text{ if and only if } Ag(L_1) \leq Ag(L_2).$$

As an example, if AVG is the arithmetic mean, then

$$L_1 \leq_{\text{AVG}} L_2 \text{ if and only if } \text{AVG}(L_1) \leq \text{AVG}(L_2).$$

### 4.4 Comparing orders of lists with the same length

In some cases, some lists orders can be compared with others.

**Proposition 1.** *Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ two lists in L, then $L_1 \leq_{MaxMin} L_2 \implies L_1 \leq_{S1to1} L_2$.*

*Proof.* Considering that $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ are two lists of degrees in L, if $L_1 \leq_{\text{MaxMin}} L_2$ then $a_i \leq b_j$ for all $i,j \in \{1,\ldots,n\}$, and consequently $a_i \leq b_i$ for all $i$, that is, $L_1 \leq_{\text{S1to1}} L_2$.

□

**Proposition 2.** *Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ two lists in L, then $L_1 \leq_{MaxMin} L_2 \implies L_1 \leq_{Ag} L_2$.*

*Proof.* Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ be two lists of degrees in L. If $L_1 \leq_{\text{MaxMin}} L_2$ then $a_i \leq b_i$ for all $i \in \{1,\ldots,n\}$. As $Ag$ is monotonic then $Ag([a_1,\ldots,a_n]) \leq Ag([b_1,\ldots,b_n])$ and $L_1 \leq_{\text{Ag}} L_2$. □

**Proposition 3.** *Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ two lists in L, if AgSym is a symmetric aggregation operator, then $L_1 \leq_{S1to1} L_2 \implies L_1 \leq_{AgSym} L_2$.*

*Proof.* Let $L_1$ and $L_2$ be two lists of degrees in L. If $L_1 \leq_{\text{S1to1}} L_2$ then $s(L_1)_i \leq s(L_2)_i$ for all $i \in \{1,\ldots,n\}$. As *AgSym* is monotonic and symmetric then $AgSym(L_1) = (AgSym([s(L_1)_1,\ldots,s(L_1)_n]) \leq AgSym([s(L_2)_1,\ldots,s(L_2)_n]) = AgSym(L_2)$ and $L_1 \leq_{\text{AgSym}} L_2$. □

It is important to notice that $\leq_{\text{AgSym}}$ is not a new preorder, but a particular case of $\leq_{\text{Ag}}$ (with *Ag* being symmetric).

If we consider Propositions 1 to 3, the following structure arises:

$$L_1 \leq_{\text{MaxMin}} L_2 \Longrightarrow \begin{cases} L_1 \leq_{\text{S1to1}} L_2 \implies L_1 \leq_{\text{AgSym}} L_2 \\ L_1 \leq_{\text{Ag}} L_2 \end{cases}$$

*Remark* 1. From the previously defined orderings, $\leq_{\text{MaxMin}}$, $\leq_{\text{S1to1}}$ and $\leq_{\text{AgSym}}$ are not affected by resorting of the compared lists, while $\leq_{\text{Ag}}$ is sensitive to resorting when *Ag* is not symmetric.

*Remark* 2. It is also important to notice that any ordering that either includes a previous sorting or is not affected by resorting, will be a preorder. For any two lists $L_1$ and $L_2$, made up of exactly the same elements but in a different order: $L_1 \leq L_2$, $L_2 \leq L_1$ and $L_1 \neq L_2$. That is, the relation will not be antisymmetric (will not be an order).

## 5 Empirical monotonicity for non-deterministic computable aggregations

How to generalize the idea of monotonicity from classical aggregation operators to non-deterministic computable aggregations is not a trivial task. In general terms, monotonicity implies that if we have two input lists $L_1 \leq L_2$, their outputs should maintain that order. Consequently, the usual concept of monotonicity of aggregation operators is not valid for non-deterministic computable aggregations, as these aggregations do not produce the same output when receiving the same input. To cope with this situation, the concept of Empirical and Theoretical Distribution of a computable aggregation ($DP^m_{L_i}$ and $\mathscr{P}(L_1)$) were previously introduced. Also, the concept of list associated to *m* executions for a given list $L_1$ ($LP(L_1)$). This concept describes the result obtained (a list $L_o \in L$) after *m* executions of the program *P* over the list $L_i$. When *m* is large enough, it allows an empirical analysis of the computable aggregation.

Beyond a probabilistic framework and modelling the output of a computable non deterministic as a distribution function (see [10]), it is possible, from an algorithmic point of view, to represent as a list of elements the output-data generated after *m* executions. The following properties try to answer the question of how to define monotony from this point of view.

Given a list of degrees $L_i = [a_1,\ldots,a_n]$, $LP^m_{L_i} = [x_1,\ldots,x_m]$ is a new list of degrees where $x_j$ is the result of the $j^{th}$ execution of $P(L_i)$, with $j = \{1,\ldots,m\}$.

**Definition 17** (Strong $\leq$-monotonicity of non-deterministic computable aggregations)**.** Let *P* be a non-deterministic computable aggregation. Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ be two lists of degrees in *L*. Let $\leq$ be a partial order (or a preorder) on the set *L* of non-empty and finite lists of degrees (see Section 4). A non-deterministic computable aggregation *P* is strong $\leq$-monotone if and only if, when $L_1 \leq L_2$ then $(LP^m_{L_1}) \leq (LP^m_{L_2})$ for any $m \geq 1$.

For example, a non-deterministic computable aggregation *P* is strong *MaxMin*-monotone, if and only if, when $L_1 \leq_{\text{MaxMin}} L_2$ then $(LP^m_{L_1}) \leq_{\text{MaxMin}} (LP^m_{L_2})$ for any $m \geq 1$.

Previous definition imposes that the orders for the input and output set of lists be the same. In the following definition we generalize this idea with two different orders.

**Definition 18** (Strong $\leq_1$-$\leq_2$-monotonicity of non-deterministic computable aggregations)**.** Let *P* be a non-deterministic computable aggregation. Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ be two lists of degrees in *L*. Let $\leq_1$ and $\leq_2$ be orders or preorders on the set *L* of non-empty and finite lists of degrees. A non-deterministic computable aggregation *P* is strong $\leq_1$-$\leq_2$-monotone, if and only if, when $L_1 \leq_1 L_2$ then $(LP^m_{L_1}) \leq_2 (LP^m_{L_2})$ for any $m \geq 1$.

**Definition 19** (Asymptotic $\leq$-monotonicity of non-deterministic computable aggregations)**.** Let $Ag : L \to [0,1]$ be a non-deterministic computable aggregation. Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ be two lists of degrees in *L*. Let $\leq$ be a partial order or preorder on the set *L* of non-empty and finite lists of degrees. A non-deterministic computable aggregation *P* is asymptotic $\leq$-monotone if and only if there exists a natural number $n_0 > 0$ such that If $L_1 \leq L_2$ then $(LP^m_{L_1}) \leq (LP^m_{L_2})$ for any $m > n_0$.

**Definition 20** (Asymptotic $\leq_1$-$\leq_2$-monotonicity of non-deterministic computable aggregations)**.** Let $Ag : L \to [0,1]$ be a non-deterministic computable aggregation. Let $L_1 = [a_1,\ldots,a_n]$ and $L_2 = [b_1,\ldots,b_n]$ be two lists of degrees in *L*. Let $\leq_1$ and $\leq_2$ be (partial) orders or preorders on the set *L* of non-empty and finite lists of degrees. A non-deterministic computable aggregation *P* is asymptotic $\leq_1$-$\leq_2$-monotone if and only if, there exists a natural number $n_0 > 0$ such that If $L_1 \leq_1 L_2$ then $(LP^m_{L_1}) \leq_2 (LP^m_{L_2})$ for any $m > n_0$.

Let us now consider how the different non-deterministic computable aggregations previously defined, behave when applying these monotonicity criteria.

**Proposition 4.**
- *The Probability sampled and k-sampled computable aggregations are Strong $\leq_{MaxMin}$-Monotonic.*
- *The Probability sampled and k-sampled computable aggregations are Strong $\leq_{MaxMin}$-$\leq_2$-Monotonic, being $\leq_2$ any of the defined orders.*

*Remark* 3. As a direct result of the definitions, Strong $\leq$-monotonicity implies Asymptotic $\leq$-monotonicity and consequently Probability sampled and *k*-sampled computable aggregations are Asymptotic $\leq_{\text{MaxMin}}$-Monotonic and Asymptotic $\leq_{\text{MaxMin}}$-$\leq_2$-Monotonic, being $\leq_2$ any of the defined orders.

Intuitively we can say that Noisy computable aggregations will have asymptotic monotonicity for all those orders that are not sensitive to resorting. But this assertion can't be proved in terms of lists comparisons, it will require further analysis based on the asymptotic behaviour related to the distribution of the values in the list. Consequently, that analysis should be considered in the framework of either the empirical distributions ($DP^m_{L_1}$) or the theoretical distributions ($\mathscr{P}(L_1)$) of the lists, being part of the open questions for future research.

## 6 Conclusions

The definition of Computable aggregation implies a rupture between functions and aggregation processes allowing the incorporation of a new class of aggregations: the non deterministic computable aggregation. This new class of computable aggregations is characterized by aggregations where the result of the process could change even if we fix the information that has to be aggregated. This situation affects the analysis of some properties as monotonicity.

In this paper, we have tried to define monotonicity for non-deterministic computable aggregations, when the program implementing the aggregation is a black box, and only input-output relations can be considered. To do so we have first characterized the output, second defined some orders to compare lists, and finally established the idea of monotonicity in terms of those previously defined orders.

In addition, the different orders defined have been compared, and the strong monotonicity of the defined non-deterministic computable aggregations have been considered. The analysis of asymptotic monotonicity should rely on the consideration of the underlying distributions, requiring a different kind of analysis to be considered for future research.

## Acknowledgement

This research has been partially supported by the Government of Spain (grant PGC2018-096509-B-I00), Comunidad de Madrid (Convenio Plurianual con la Universidad Politécnica de Madrid en la línea de actuación Programa de Excelencia para el Profesorado Universitario, and grant S2013/ICCE-2845), Complutense University (UCM Research Group 910149) and Universidad Politécnica de Madrid.

## References

[1] G. Beliakov, D. Gómez, S. James, . Montero, J. Rodríguez, Approaches to learning strictly-stable weights for data missing values, Fuzzy Sets and Systems 325 (2017) 97–113.

[2] G. Beliakov, A. Pradera, T. Calvo, Aggregations Functions: A guide for Practitioners, Springer, 2007.

[3] B. Bouchon-Meunier, Aggregation and fusion of imperfect information, Vol. 12, Physica, 2013.

[4] H. Bustince, F. Herrera, J. Montero (Eds.), Fuzzy Sets and Their Extensions: Representation, Aggregation and Models, Vol. 220 of Studies in Fuzziness and Soft Computing, Springer-Verlag, Berlin Heidelberg, 2008.

[5] T. Calvo, A. Kolesárová, M. Komorníková, R. Mesiar, Aggregation operators: properties, classes and constructions methods, in: C. T., M. G., M. R. (Eds.), Aggregation Operators, Vol. 97 of Studies in Fuzziness and Soft Computing, Springer, 2002, pp. 3–104.

[6] L. Garmendia, D. Gómez, L. Magdalena, J. Montero, Analyzing non-deterministic computable aggregations, in: M.-J. Lesot, S. Vieira, M. Z. Reformat, J. P. Carvalho, A. Wilbik, B. Bouchon-Meunier, R. R. Yager (Eds.), Information Processing and Management of Uncertainty in Knowledge-Based Systems, Springer International Publishing, Cham, 2020, pp. 551–564.

[7] D. Gómez, J. Montero, A discussion on aggregations operators, Kybernetika 40 (2004) 107–120.

[8] D. Gómez, K. Rojas, J. Montero, J. Rodríguez, G. Beliakov, Consistency and stability in aggregation operators, an application to missing data problems, International Journal of Computational Intelligence Systems 7 (2014) 595–604.

[9] L. Magdalena, L. Garmendia, D. Gómez, R. G. del Campo, J. T. Rodríguez, J. Montero, Types of recursive computable aggregations, in: 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2019, pp. 1–6.

[10] L. Magdalena, L. Garmendia, D. Gómez, J. Montero, Population monotonicity of non-deterministic computable aggregations, in: 2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2021.

[11] M. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, 1967.

[12] J. Montero, R. G. del Campo, L. Garmendia, D. Gómez, J. Rodríguez, Computable aggregations, Information Sciences 460 (2018) 439–449.

[13] P. Olaso, K. Rojas, D. Gómez, J. Montero, A generalization of stability for families of aggregation operators, Fuzzy Sets and Systems.

[14] K. Rojas, D. Gómez, J. Montero, J. Rodríguez, Strictly stable families of aggregation operators, Fuzzy Sets and Systems 228 (2013) 44–63.

[15] N. Wirth, Algorithms + Data Structures = Programs, Prentice-Hall, 1976.