Complexity of Increasing ϕ-Recursive Computable Aggregations

Ramón González-del-Campo
Faculty of Informatics, Complutense University of Madrid
Email: rgonzale@ucm.es

Luis Garmendia
Faculty of Informatics, Complutense University of Madrid
Email: lgarmend@fdi.ucm.es

Javier Montero
Faculty of Mathematics, Complutense University of Madrid
Email: monty@mat.ucm.es

***Abstract*—In this paper the new concepts of $\mathcal{O}(f(n))$-increasing ϕ-recursive and $\mathcal{O}(f(n))$-decreasing ψ-recursive computable aggregation and expansion function are proposed to describe the computational cost of recursive computational aggregations when the universe of the discourse is increased or decreased are related. The complexity costs of the expansion functions with the complexity costs of its recursive computational aggregations.**

I. Introduction

As stressed in [11], much effort is needed in analyzing the properties of the algorithms we apply to solve aggregation problems in practice. In fact, in [11] , the authors pointed out that it is the avaliable algorithm what defines each aggregation, making feasible a specific solution to each possible problem, of course depending on decision maker's tools and capacities. Such computable aggregations come with a protocol that enables us to face aggregation problems in a specific general framwork. For example, when the cardinal of data is not being fixed, or cannot be a priori fixed. In particular, recursive aggregations [4], [5], [6], [7], [9] are a special kind of computable aggregations that play a strong role when the universe is modified. The computational cost is critical when it is necessary to process a huge amount of data.

When recursive aggregations are used it is necessary to know the cost to recompute the new value of the aggregation if the universe is changed adding or removing data. In this paper, two new concepts are proposed to describe the computational cost of recursive computational aggregations when the universe of the discourse is modified by adding or removing an element. On the one hand, the $\mathcal{O}(f(n))$-increasing ϕ-recursive computable aggregations is a set of recursive computable aggregations that have a computational cost bounded by $f(n)$ when an element is added to the universe. The expansion function of a computable recursive aggregation allows to know its computational cost. On the other hand, the $\mathcal{O}(f(n))$-decreasing ψ-recursive computable aggregations are defined in similar way when an element is removed from the universe.

II. Preliminaries

The concept of computational complexity cost of an algorithm is an important consideration in computer sciences as a degree to measure the quality and usability of programs. It can be measured in terms of time or memory usage, but it is usually measured in terms of number of operations, and how this number grows as the size of data grows comparing with a function or order in which adding constants, multipliers or lower order functions do no affect the main kind of growing of the higher order.

Definition 1. *[3] Let $f : \mathbb{N} \to \mathbb{R}$ be a function. The set of functions in the order of f, $\mathcal{O}(f)$, are defined as follows:*

$$\mathcal{O}(f) \equiv \{g : \mathbb{N} \to \mathbb{R} \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N}\ \forall n \geq n_0 \\ g(n) \leq cf(n)\}$$

The order of f contains all the functions that grows up slower than f.

Definition 2. *[3] Let $f : \mathbb{N} \to \mathbb{R}$ be a function. The computational complexity of f, $\Theta(f)$, is defined as follows:*

$$\Theta(f) \equiv \{g : \mathbb{N} \to \mathbb{R} \mid \exists c, d \in \mathbb{R}, n_0 \in \mathbb{N}\ \forall n \geq n_0 \\ df(n) \leq g(n) \leq cf(n)\}$$

Definition 3. *[3] Let $f, g : \mathbb{N} \to \mathbb{R}$ be two functions. It is said f has a complexity lower than g if $\mathcal{O}(f) \subset \mathcal{O}(g)$*

Proposition 1. *[3] Let q, a be two real numbers such that $q > 1$ and $a > 1$. Then:*

$$\mathcal{O}(1) \subset \mathcal{O}(log(n)) \subset \mathcal{O}(n) \subset \mathcal{O}(n^q) \subset \mathcal{O}(a^n) \subset O(n!)$$

In the following definition the most usual types of complexity are introduced.

Definition 4. *[3] Let $g : \mathbb{N} \to \mathbb{R}^+$ be a function. Then:*

- g has constant complexity if g belongs to $\Theta(1)$, i.e, if g grows up as fast as $f(n) = 1$.
- g has logarithmic complexity if g belongs to $\Theta(log(n))$, i.e, if g grows up as fast as $f(n) = log(n)$.
- g has linear complexity if g belongs to $\Theta(n)$, i.e, if g grows up as fast as $f(n) = n$.
- g has polynomial complexity if g belongs to $\Theta(n^q)$ for some $q > 1$, i.e, if g grows up as fast as $f(n) = n^q$.
- g has exponential complexity if g belongs to $\Theta(a^n)$ for some $a > 1$, i.e, if g grows up as fast as $f(n) = a^n$.

- g has factorial complexity if g belongs to $\Theta(n!)$, i.e, if g grows up as fast as $f(n) = n!$.

Definition 5. *[3] The computational complexity cost of an algorithm is the order of the function that gives the computing time of the algorithm.*

It is possible a definition of computational complexity cost focusing on the number of operations to complete the algorithm:

Definition 6. *[3] The computational complexity cost of an algorithm is the order of the function that gives a bound for the number of operations of the algorithm.*

Definition 7. *[1] A L list is an Abstract Data Type (ADT) that represents a sequence of values. A list can be defined by its behavior and its implementation must provide at least the following operations:*

- *Test whether a list is empty or not.*
- *Add a value.*
- *Remove a value.*
- *Compute the length (number of values) of a list.*

A list can be defined under a template data. For example, a list $L < [0,1] >$ is a list of values in $[0,1]$.

Talking about complexity of algorithms implies to show the code of them. There are a lot of programming languages (C++, Python, Java,...). Python is a easy to understand programming language. Even if you do not know Python, you can understand a program written in Python. That is the reason why the programs are written in Python in this paper. All programs here showed can be rewritten in any other programming language.

Definition 8. *[5] A left-recursive connective rule is a family of connective operators:*

$$(Ag : [0,1]^n \to [0,1])_{n>1}$$

such that there exists a sequence of binary operators:

$$(L_n : [0,1]^2 \to [0,1])_{n>1}$$

verifying:

- $Ag(a_1, a_2) = L_2(a_1, a_2)$
- $Ag(a_1 \ldots, a_n) = L_n(Ag(a_1, \ldots, a_{n-1}), a_n)$ *for all* $n > 2$

for some ordering rule π.

In similar way a right-recursive connective rule can be defined.

A right-recursive connective and left-recursive connective rule is called recursive connective rule.

Definition 9. *[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.*

There exists some other proposals to fusion information as the pre-aggreation functions that introduce the concept of directional monotonicity and have been useful in some applications.

Definition 10. *[10] A mapping $F : [0,1]^n \to [0,1]$ is a n-dimensional pre-aggregation function if it satisfies:*

- *There exists a real vector $r \in [0,1]^n$ with $r \neq 0$ such that is r-growing.*
- *$F(0, \ldots, 0 = 0)$ and $F(1, \ldots, 1 = 1)$.*

Next definition shows a wider point of view about aggregation. It is possible to extend the domain of aggregations to lists of elements L with generic types T. For example, T can be an image and the aggregation that process it can make the fussion of images.

In this paper of article, we will focus on the aggregations that can be computed using a program and the cost of computation of these aggregations.

Definition 11. *[11] Let $L < T >$ be a list of n elements of type T. A computable aggregation Ag_c is a program P that transforms the list $L < T >$ into an element of T.*

Definition 12. *[8] Let $L = \{x_1, \ldots, x_n\}$ be a list of values. A computable aggregation rule Ag_c is recursive if there exists a mapping $\phi : [0,1]^2 \to [0,1]$ such that:*

$$Ag_c(L) = \begin{cases} x_1, & \text{if } lenght(L) = 1; \\ \phi(Ag_c(x_1, ..., x_{n-1}), x_n), & \text{if } lenght(L) > 1. \end{cases}$$

Definition 13. *[8] A computation aggregation rule is expansible if there exists a mapping $\phi : [0,1]^2 \to [0,1]$ satisfying the following property:*

$$Ag_c(L_1 \cup L_2) = \phi(Ag_c(L_1), Ag_c(L_2))$$

where ϕ is an algorithm with linear or lower computational complexity cost.

Note. Let $Comp$, Rec and Exp be the computable aggregations, the recursive aggregations and the expansible aggregations respectively. Then:

$$Exp \subset Rec \subset Comp$$

Talking about complexity of algorithms implies to show the code of algorithms that implement them. There are a lot of programming languages (C++, Python, Java,...). Python is a easy to understand programming language. Even if you do not know Python, you can understand a program written in Python. That is the reason why the programs are written in Python in this paper. All programs here showed can be rewritten in any other programming language.

III. Increasing ϕ-Recursive Computable Aggregations Rules

Let $L = \{x_1, \ldots, x_n\}$ be a list of values and n its length.

Definition 14. *A computable aggregation rule Ag_c is $\mathcal{O}(f(n))$-increasing ϕ-recursive if there exists a mapping $\phi : [0,1]^2 \times L \to [0,1]$ such that:*

- $Ag_c(L) = \begin{cases} x_1, & \text{if } n = 1; \\ \phi(Ag_c(x_1, ..., x_{n-1})), x_n, \{x_1, \ldots, x_{n-1}\}), & \text{if } n > 1. \end{cases}$
- *ϕ has a $\mathcal{O}(f(n))$ complexity cost.*

Definition 15. *A $\mathcal{O}(f(n))$-increasing ϕ-recursive computable aggregation rule Ag_c is non depending of length L if there exists a mapping $\phi : [0,1]^2 \to [0,1]$ such that:*

- $Ag_c(L) = \begin{cases} x_1, & \text{if } n = 1; \\ \phi(Ag_c(x_1, ..., x_{n-1})), x_n), & \text{if } n > 1. \end{cases}$
- *ϕ has a $\mathcal{O}(f(n))$ complexity cost.*

Next lemmas relate types of increasing ϕ-recursive computable aggregations with recursive computable aggregations and explansible computable aggregations given in Definition 12 and Definition 13.

The next two lemmas are trivial:

Lemma 1. *A recursive computable aggregation is $\mathcal{O}(f(n))$-increasing ϕ-recursive for some $f(n)$.*

Lemma 2. *If Ag_c is an explansible computable aggregation, then Ag_c is a $\mathcal{O}(n)$-increasing ϕ-recursive computable aggregation.*

The following lemmas show some $\mathcal{O}(n)$-increasing ϕ-recursive computable aggregations.

Lemma 3. *Arithmetic mean is a $\mathcal{O}(1)$-increasing ϕ-recursive computable aggregation.*

Proof. $Ag_c(x_1, \ldots, x_{n+1}) =$
$= \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n+1}(\sum_{i=1}^{n} x_i + x_{n+1}) = \frac{n}{n+1}\sum_{i=1}^{n} x_i + \frac{n}{n+1}x_{n+1} = \frac{n}{n+1}Ag_c(x_1, \ldots, x_n) + \frac{n}{n+1}x_{n+1}$

So $\phi(x, y, n) = \frac{n}{n+1}x + \frac{1}{n+1}y$.

The next two programs compute $\phi(x, y, \{x_1, \ldots, x_n\})$ when n (length of $\{x_1, \ldots, x_n\}$) is known and when n is not known:

- n is known:

```
def phi(x,y,n):
    return (n*x+y)/(n+1)
```

So ϕ is $\mathcal{O}(1)$ complexity.

- n is unknown:

```
def phi(x,y,l):
    n=len(l)
    return (n*x+y)/(n+1)
```

Then, it is $\mathcal{O}(n)$ complexity to compute *length(l)*.

□

Lemma 4. *The Product computable aggregation is a $\mathcal{O}(1)$-increasing ϕ-recursive computable aggregation.*

Proof. $Ag_c(x_1, \ldots, x_{n+1}) =$
$= \prod_{i=1}^{n+1} x_i = x_{n+1} * \prod_{i=1}^{n} x_i$

So $\phi(x, y) = x * y$, which have a $\mathcal{O}(1)$ constant complexity cost.

The next program computes $\phi(x, y)$:

```
def phi(x,y):
    return x*y
```

So ϕ is $\mathcal{O}(1)$ complexity. □

Lemma 5. *Bounded sum ($\min\{1, \sum_{i=1}^{n} x_i\}$) is a $\mathcal{O}(1)$-increasing ϕ-recursive computable aggregations.*

Proof. There exist three cases:

1) If $Ag_c(x_1, \ldots, x_n) = 1$, then $Ag_c(x_1, \ldots, x_{n+1}) = 1$
2) If $Ag_c(x_1, \ldots, x_n) < 1$ and $Ag_c(x_1, \ldots, x_n) + x_{n+1} \geq 1$, then $Ag_c(x_1, \ldots, x_{n+1}) = 1$
3) If $Ag_c(x_1, \ldots, x_n) + x_{n+1} < 1$, then $Ag_c(x_1, \ldots, x_{n+1}) = Ag_c(x_1, \ldots, x_n) + x_{n+1}$

The next program computes $\phi(x, y)$:

```
def phi(x,y):
    if x==1:
        res=1
    else:
        if x+y>=1:
            res=1
        else:
            res x+y
    return res
```

with $\mathcal{O}(1)$ constant complexity cost. □

Lemma 6. *Geometric mean is a $\mathcal{O}(1)$-increasing ϕ-recursive computable aggregations if n is known.*

Proof. $Ag_c(x_1, \ldots, x_{n+1}) =$
$= (\prod_{i=1}^{n+1} x_i)^{\frac{1}{n+1}} = (x_n \prod_{i=1}^{n} x_i)^{\frac{1}{n+1}} = (x_n)^{\frac{1}{n+1}}(\prod_{i=1}^{n} x_i)^{\frac{1}{n+1}} = (x_n)^{\frac{1}{n+1}}((\prod_{i=1}^{n} x_i)^{\frac{1}{n}})^{\frac{n}{n+1}}$
$= (x_n)^{\frac{1}{n+1}} Ag_c(x_1, \ldots, x_n))^{\frac{n}{n+1}}$

So $\phi(x, y) = x^{\frac{n}{n+1}} * y^{\frac{1}{n+1}}$.

The two next programs compute $\phi(x, y, \{x_1, \ldots, x_n\})$ depending if n (length of $\{x_1, \ldots, x_n\}$) is known or not:

- n is known:

```
def phi(x,y,n):
    return res x**(n/(n+1))*y**(n/(n+1))
```

So ϕ is $\mathcal{O}(1)$ complexity.

- n is unknown:

```
def phi(x,y,l):
    n=len(l)
    return res x**(n/(n+1))*y**(n/(n+1))
```

So ϕ is $\mathcal{O}(n)$ complexity cost.

□

Lemma 7. *The forward and backward aggregations (Ag^f_{cn} and Ag^b_{cn}) over the binary operator A are $\mathcal{O}(c(x, y))$-increasing ϕ-recursive computable aggregations.*

Proof. Let $A(x, y)$ be a binary operator and let $c(x, y)$ be its complexity.

$Ag^f_{n+1}(\{x_1, \ldots, x_{n+1}\}) = A(Ag^f_{cn}(\{x_1, \ldots, x_n\}, x_{n+1})$.

Then $\phi(x, y) = A(x, y)$ and $\phi(x, y)$ is computed with $\mathcal{O}(c(x, y))$. So Ag^f_{cn} is a $\mathcal{O}(c(x, y))$-increasing ϕ-recursive computable aggregation.

In similar way it is possible to prove Ag^b_{cn} is a $\mathcal{O}(c(x, y))$-increasing ϕ-recursive computable aggregation.

□

Corollary 1. *Minimum, Maximum, Product, Forward and Backward aggregations are non depending of length increasing ϕ-recursive computable aggregations.*

Aggregation	ϕ	Complexity of ϕ
Arithmetic mean	$\frac{n}{n+1}x + \frac{1}{n+1}y$	$\mathcal{O}(1)$,$\mathcal{O}(n)$
Minimum	$\min\{x, y\}$	$\mathcal{O}(1)$
Maximum	$\max\{x, y\}$	$\mathcal{O}(1)$
Product	$x * y$	$\mathcal{O}(1)$
Bounded sum	See program	$\mathcal{O}(1)$
Geometric mean	$x^{\frac{n}{n+1}} * y^{\frac{1}{n+1}}$	$\mathcal{O}(1)$,$\mathcal{O}(n)$
$Ag^f_{cn}(\{x_1, \ldots, x_n\})$	$A(x, y)$	$\mathcal{O}(c(x, y))$
$Ag^b_{cn}(\{x_1, \ldots, x_n\})$	$A(x, y)$	$\mathcal{O}(c(x, y))$

TABLE I
EXPANSION FUNCTIONS COMPLEXITY COST OF SOME INCREASING ϕ-RECURSIVE COMPUTABLE AGGREGATIONS.

Proof. Trivial. □

Lemma 8. *Let $Ag_c^{\mathcal{O}(f(n))}$ and $Ag_c^{\mathcal{O}(g(n))}$ be the sets of $\mathcal{O}(f(n))$-increasing ϕ-recursive and $\mathcal{O}(g(n))$-increasing ϕ-recursive computable aggregations respectively. If $f(n)$ belongs to $\mathcal{O}(g(n))$, then*

$$Ag_c^{\mathcal{O}(f(n))} \subseteq Ag_c^{\mathcal{O}(g(n))}$$

Proof. Trivial. □

Theorem 1. *Let ϕ be the expansion function of Ag_c. If the complexity of ϕ is $\Theta(f(n))$ then Ag_c is approachable with complexity $\Theta(n * f(n))$.*

Proof. Let $f(n)$ be the function that represents the computing time of ϕ. The next algorithm computes $Ag_c(x_1, \ldots, x_n)$ using ϕ:

```
def phi(y,x):
...
def Ag_c(psi,l):
    for x in l:
        aux=phi(aux,x)
    return aux
```

The number of times that the code of the loop `for` is executed depends on the length of list `l`, n. So Ag_c takes a computing time $n * f(n)$. □

Corollary 2. *If ϕ has a polynomial complexity cost, then Ag_c has a polynomial complexity cost.*

Proof. If ϕ has a polynomial complexity its computing time is $f(n) = k * n^a$ for some a. Then, due to Theorem 1 the computing time for Ag_c using ϕ is $k' * n * k * n^a = k'' n^{a+1}$. So the computing time of Ag_c belongs to $\Theta(n^{a+1})$ which is also polynomial complexity cost. □

IV. Decreasing ψ-Recursive Computable Aggregations Rules

Let Ω be a region in $\mathbb{R}^2$ such that $\Omega \subseteq \mathbb{R}^2$.

Definition 16. *Let $L = \{x_1, \ldots, x_n\}$ be a list of values. A computable aggregation rule Ag_c is $\mathcal{O}(f(n))$-decreasing ψ-recursive in Ω if there exists a mapping $\psi : [0, 1]^2 \to [0, 1]$ such that:*

- *For all (x, y) in Ω: $Ag_c(L \setminus \{x_n\}) = \psi(Ag_c(x_1, ..., x_n)), x_n)$ if $lenght(L) > 1$*
- *ψ has a $\mathcal{O}(f(n))$ complexity cost.*

Lemma 9. *Arithmetic mean is a $\mathcal{O}(1)$-decreasing ψ-recursive computable aggregation.*

Proof. $Ag_c(x_1, \ldots, x_n) =$
$= \frac{1}{n}\sum_{i=1}^{n} x_i = = \frac{1}{n}(\sum_{i=1}^{n-1} x_i + x_n) = \frac{1}{n}\sum_{i=1}^{n-1} x_i + \frac{1}{n} * x_n = \frac{n-1}{n} Ag_c(x_1, \ldots, x_{n-1}) + \frac{1}{n} * x_n$

So $Ag_c(x_1, \ldots, x_{n-1}) = \frac{n}{n-1}(Ag_c(x_1, \ldots, x_n) - \frac{1}{n-1}x_n)$ and $\psi(x, y, n) = \frac{n}{n-1}x + \frac{1}{n-1}y$ in $\Omega = \mathbb{R}^2$.

The next program compute $\psi(x, y, \{x_1, \ldots, x_n\})$ depending if n (length of $\{x_1, \ldots, x_n\}$) is known or not:

- n is known:

```
def phi(x,y,n):
    return n/(n-1)*x-1/(n-1)y
```

So ψ is $\mathcal{O}(1)$ complexity.

- n is unknown:

```
def phi(x,y,l):
    n=len(l)
    return n/(n-1)*x-1/(n-1)y
```

So ψ has $\mathcal{O}(n)$ complexity for the computation of *length(L)*.

□

Lemma 10. *Minimum and Maximum are $\mathcal{O}(1)$-decreasing ψ-recursive computable aggregations.*

Proof. If $\min\{x_1, \ldots, x_n\} < x_n$, then $\min\{x_1, \ldots, x_{n-1}\} = \min\{x_1, \ldots, x_n\}$ and $Ag_c(x_1, \ldots, x_{n-1}) = Ag_c(x_1, \ldots, x_n)$. So $\Omega : \{(x, y) : x < y\}$

The next program computes $\psi(x, y)$:

```
def psi(x,y):
    return x
```

The complexity of $\psi(x, y, l)$ is $\mathcal{O}(1)$ in Ω.

Similar considerations can be done for Maximun. □

Lemma 11. *Product is a $\mathcal{O}(1)$-decreasing ψ-recursive computable aggregation.*

Proof. $Ag_c(x_1, \ldots, x_n) =$
$= \prod_{i=1}^{n} x_i = x_n * \prod_{i=1}^{n-1} x_i$

So $Ag_c(x_1, \ldots, x_n) = x_n * Ag_c(x_1, \ldots, x_{n-1})$ and $\psi(x, y) = x/y$.

The next program computes $\psi(x, y)$:

```
def psi(x,y):
    return x/y
```

So ψ has $\mathcal{O}(1)$ constant complexity cost. □

Lemma 12. *Geometric mean is a $\mathcal{O}(1)$-decreasing ψ-recursive computable aggregations if n is known.*

Proof. $Ag_c(x_1, \ldots, x_{n-1}) =$
$= (\prod_{i=1}^{n-1} x_i)^{\frac{1}{n-1}} = \frac{x_n}{x_n}((\prod_{i=1}^{n-1} x_i)^{\frac{1}{n-1}})^{\frac{n}{n}} = \frac{1}{x_n}((\prod_{i=1}^{n} x_i)^{\frac{1}{n}})^{\frac{n}{n-1}} = \frac{1}{x_n} Ag_c(x_1, ..., x_n)^{\frac{n}{n-1}}$

So $\psi(x, y, n) = \frac{1}{y}x^{\frac{n}{n-1}}$ □

The next program computes $\psi(x, y, n)$:

Aggregation	ψ	Complexity of ψ	Ω
Arithmetic mean	$\frac{n}{n-1}x + \frac{1}{n-1}y$	$\mathcal{O}(1)$,$\mathcal{O}(n)$	$\mathbb{R}^2$
Product	x/y	$\mathcal{O}(1)$	$\mathbb{R}^2$
Geometric mean	$\frac{1}{y}x^{\frac{n}{n-1}}$	$\mathcal{O}(1)$,$\mathcal{O}(n)$	$\mathbb{R}^2$
Minimum	x	$\mathcal{O}(1)$	$x < y$
Maximum	x	$\mathcal{O}(1)$	$x > y$
Bounded sum	$x - y$	$\mathcal{O}(1)$	$x < 1$

TABLE II
EXPANSION FUNCTIONS COMPLEXITY COST OF SOME DECREASING ψ-RECURSIVE COMPUTABLE AGGREGATIONS.

```
def phi(x,y,n):
    return res x**(n/(n-1))/y
```

So ψ has $\mathcal{O}(1)$ constant complexity cost if n is known.

Lemma 13. *Bounded sum* $(\min\{1, \sum_{i=1}^{n} x_i\})$ *is a* $\mathcal{O}(1)$*-decreasing* ψ*-recursive computable aggregations.*

Proof. If $Ag_c(x_1, \ldots, x_n) < 1$, then $Ag_c(x_1, \ldots, x_{n-1}) = Ag_c(x_1, \ldots, x_n) - x_n$. So $\Omega : \{(x, y) : x < 1\}$

The next program computes $\psi(x, y)$:

```
def phi(x,y):
    if x<1:
        res=x-y
    return res
```

So ψ has $\mathcal{O}(1)$ constant complexity cost. □

Lemma 14. *Let* $Ag_{c\mathcal{O}(f(n))}$ *and* $Ag_{c\mathcal{O}(g(n))}$ *be the sets of* $\mathcal{O}(f(n))$*-decreasing* ψ*-recursive and* $\mathcal{O}(g(n))$*-decreasing* ψ*-recursive computable aggregations respectively. If* $f(n)$ *belongs to* $\mathcal{O}(g(n))$*, then*

$$Ag_{c\mathcal{O}(f(n))} \subseteq Ag_{c\mathcal{O}(g(n))}$$

Proof. Trivial. □

V. Conclusions

The main goal in this paper is to study the behaviour of computable recursive aggregations when the universe of discourse is changed. It is possible to classify computable recursive aggregations by the complexity of their expansion and the reduction function complexity cost. Moreover, it is found a relation between the complexity of a computable recursive aggregation and its expansion function cost complexity.

Acknowledgment

This research has been partially supported by the Government of Spain (grant TIN2015-66471-P), the Government of Madrid (grant S2013/ICE-2845) and Complutense University (UCM Research Group 910149).

References

[1] G. Barnett and L. Del Tonga. *Data Structures and Algorithms*. DotNetSlackers, 2008.
[2] G. Beliakov, A. Pradera and T. Calvo. *Aggregations Functions: A guide for Practitioners*. Springer, 2007.
[3] B. Brassard. *Fundamentals of Algorithmics*. Pearson, 2015.
[4] H. Bustince, B. De Baets, J. Fernandez, R. Mesiar and J. Montero. A generalization of the migrativity property of aggregation functions. *Information Sciences*, 191:76 – 85, 2012.
[5] V. Cutello and J. Montero. Recursive connective rules. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2(14):3–20, 1999.
[6] A. del Amo, J. Montero and E. Molina. Representation of consistent recursive rules. *European Journal of Operational Research*, 130(1):29–53, 2001.
[7] D. Gómez and J. Montero. A discussion on aggregations operators. *Kybernetika*, 40:107–120, 2004.
[8] R. González del Campo, L. Garmendia and J. Montero. Expansible computable aggregation rules. In *Proceedings of the 2015 International Conference on Intelligent Systems and Knowledge Engineering, ISKE 2015, Taipei, Taiwan*, pages 8–11, November 2015.
[9] A. Kolesárová, R. Mesiar and J. Montero. Sequential aggregation of bags. *Information Sciences*, 294:305–314, 2015.
[10] J. Giancarlo Lucca, G. Pereira Dimuro, B. R. C. Bedregal, R. Mesiar, A.árová and H. Bustince. Preaggregation functions: Construction and an application. *IEEE Trans. Fuzzy Systems*, 24(2):260–272, 2016.
[11] J. Montero, R. González del Campo, L. Garmendia, D. Gómez and J. Tinguaro. Computable aggregations. *Information Sciences*, 2, 2017.