mathematics

MDPI

Article

An Algorithm for Computing All Rough Set Constructs for Dimensionality Reduction

Yanir González-Díaz [1,*], José Fco. Martínez-Trinidad [1,†], Jesús A. Carrasco-Ochoa [1,†] and Manuel S. Lazo-Cortés [2,†]

[1] Department of Computer Science, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla 72840, Mexico; fmartine@inaoep.mx (J.F.M.-T.); ariel@inaoep.mx (J.A.C.-O.)
[2] Graduate Division, Tecnológico Nacional de México/IT Tlalnepantla, Tlalnepantla de Baz 54070, Mexico; manuel.lc@tlalnepantla.tecnm.mx
* Correspondence: ygdiaz@inaoep.mx
† These authors contributed equally to this work.

Abstract: In rough set theory, a construct is an attribute subset with the same ability to discern objects belonging to different classes as the whole set of attributes, while maintaining the similarity between objects belonging to the same class. Although algorithms for reducts computation can be adapted to compute constructs, practical problems exist where these algorithms cannot compute all constructs within a reasonable time frame. Therefore, this paper introduces an algorithm for computing all constructs of a decision system. The results of experiments with various decision systems (both artificial and real-world) suggest that our algorithm is, in most cases, faster than the state-of-the-art algorithms when the simplified binary discernibility–similarity matrix has a density of less than 0.29.

Keywords: rough sets; constructs; fast algorithms; feature selection

MSC: 68T37; 68T30; 68Q25; 68T10

Citation: González-Díaz, Y.; Martínez-Trinidad, J.F.; Carrasco-Ochoa, J.A.; Lazo-Cortés, M.S. An Algorithm for Computing All Rough Set Constructs for Dimensionality Reduction. *Mathematics* **2024**, *12*, 90. https://doi.org/10.3390/math12010090

Academic Editors: Chengyou Wang, Xiao Zhou, Zhaobin Wang and Yingchun Guo

Received: 10 October 2023
Revised: 14 November 2023
Accepted: 16 November 2023
Published: 27 December 2023

1. Introduction

In rough set theory (*RST*) [1], constructs [2], like reducts [3], are subsets of attributes that allow discerning objects belonging to different classes to at least the same extent as the complete set of attributes, but unlike reducts, constructs also maintain similarity between objects belonging to the same class. Thus, although constructs are commonly larger than reducts, they are a better option for building a reduced representation of a decision system preserving as much knowledge as possible. Constructs can be used to build rule-based classifiers [4], as feature selectors [5], in problems of reduction of the representation space of the objects [6], among others.

The most challenging problem related to constructs involves computing all constructs of a decision system, which is an NP-hard problem (reducible to the Boolean satisfiability problem (SAT) in polynomial time). On the other hand, computing constructs has been little explored; only a few studies have proposed methods to compute constructs using algorithms designed for reducts computation [7,8]. The study presented in [7] is focused on computing constructs using algorithms that compute reducts working on the discernibility function. In contrast, the study presented in [8] focuses on computing constructs using algorithms for reducts computation that operate on the binary discernibility matrix.

Although both works introduce methods that allow computing all constructs using algorithms designed to compute reducts, it is worth mentioning that, for reducts computation, in most datasets, it is better to operate with the binary discernibility matrix rather than the discernibility function, since by considering specific properties of Boolean operations and their bitwise implementations, faster executions of the algorithms can be achieved [9,10].

Because of the above-mentioned, an algorithm for computing all constructs in a dataset, based on the method described in [8], is introduced in this paper. The algorithm introduced in this work for ***c***omputing all ***c***onstructs uses the concepts of ***g***ap elimination and ***a***ttribute ***c***ontribution (denoted as ***CC-GEAC***), which is also used in the most successful algorithms for reducts computation reported in the literature [10–12]. The *CC-GEAC* algorithm introduces a pruning strategy that allows for the a priori discarding of some attribute subsets. This pruning strategy avoids unnecessary evaluations conducted during the computation of constructs with the most successful algorithms for reducts computation: fast-CT_EXT [11], fast-BR [12], and GCreduct [10] following the method described in [8], as will be demonstrated later. Based on our experiments, it is observed that *CC-GEAC*, in most cases, evaluates fewer candidates than the three algorithms above when they calculate all the constructs following [8].

In summary, the contributions of this paper are as follows:

A new algorithm based on gap elimination and attribute contribution is proposed for computing all constructs in a binary matrix calculated from a dataset, as described in [8].

This algorithm includes a pruning strategy that allows for the a priori discarding of more attribute subsets than other pruning strategies (also based on gap elimination and attribute contribution) used in state-of-the-art algorithms.

The rest of this document is organized as follows: Section 2 reviews the related works. In Section 3, basic concepts are presented. The *CC-GEAC* algorithm is introduced in Section 4. The experiments and results are shown in Section 5, and finally, in Section 6, our conclusions and some directions for future work are provided.

2. Related Works

Constructs computation has been little explored, as mentioned earlier. This section reviews the works reported in the literature on this issue.

In [13], the concept of a construct is introduced, and the relation between reducts and constructs is studied. In this work, Susmaga mentioned that constructs could be computed using a proper modification of an algorithm for reducts computation. This work adapted the fast reducts generating algorithm (*FRGA*) proposed in [14] to compute constructs.

In [7], a new method for computing constructs of a *DS* is introduced. The method consists of processing different parts of the object's pairwise comparison matrix (*PCM*) [15] to generate inter-class reducts, intra-class reducts, and constructs. This method has two phases: in the first phase, empty, repeated, and non-minimal elements (attribute subsets) with respect to inclusion are eliminated from the PCM; from this phase, the sorted absorbed pairwise comparison list $(SAPCL)$ is obtained. In the second phase, the SAPCL is provided as the input to an algorithm for reducts computation. The result of the second phase constitutes the output of the algorithm (i.e., the set of reducts/constructs). The particular type of output is determined by the type of PCM supplied as input to the first phase of the algorithm [7]. This method allows computing constructs using algorithms for reducts computation as long as these algorithms work on the discernibility function.

In [8], the relationship between constructs, reducts, and irreducible testors is studied, showing how algorithms for computing irreducible testors, designed into the logical combinatorial pattern recognition approach [16], can be used to compute constructs. Although the authors did not propose a new algorithm, this work increases the set of available algorithms for reducts computation that can be used for computing constructs, since the authors provide a way to compute constructs by defining a new binary matrix, where algorithms to compute reducts and irreducible testors can be used for computing constructs. This new matrix is constructed by comparing all pairs of objects in different ways, depending on whether or not they belong to the same class, unlike the binary discernibility matrix that only considers pairs of objects belonging to different classes. In this way, when two objects from different classes are compared, if an attribute distinguishes these objects, a 1 in the corresponding matrix entry is put. It means that this attribute should be taken into account when a construct is built. Otherwise, when the objects under comparison belong to the same

class, if an attribute does not distinguish these objects, a 1 is placed in the corresponding matrix entry. This is because the attribute contributes to preserving the similarity between the objects. This means that this attribute should be taken into account when a construct is built. As shown in [17], reducts and irreducible testors (*IT*) are closely related concepts, and under certain conditions, they coincide. Therefore, constructs can be computed by methods for computing reducts and irreducible testors by using this new binary matrix.

Even though either method, [7] or [8], could be used to compute constructs, the problem of finding all constructs has been little explored. This justifies the search for more efficient algorithms to compute constructs.

3. Basic Concepts

In RST, the data in a supervised classification problem are represented by a decision system (*DS*). Formally, a *DS* can be represented by a pair $DS = \langle \mathcal{O}, \mathcal{Q} \rangle$, where:

- $\mathcal{O}$ represents a finite non-empty set of objects $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ called universe;
- $\mathcal{Q}$ denotes a finite non-empty set of attributes, such that $\mathcal{Q} = \mathcal{C} \cup \{d\}$ and $\mathcal{C} \cap \{d\} = \emptyset$; being $\mathcal{C} \neq \emptyset$ the set of condition attributes and d the decision attribute.

For every attribute $a \in \mathcal{Q}$, there is an information function $I_a : \mathcal{O} \to V_a$, where V_a is a finite non-empty set of values for the attribute a, and $I_a(o_i)$ denotes the value corresponding to the object o_i in the attribute a.

A *DS* is commonly represented as a matrix, in which columns are associated with attributes, rows to objects, and cells to attribute values of objects. A *DS* is considered consistent if any combination of values from the entire set of condition attributes defines the value of the decision attribute; otherwise, it is considered inconsistent.

To express the fact that a set of attributes cannot discern objects in a *DS*, the *indiscernibility relation* [1], is defined for a non-empty set of attributes $\mathcal{P} \subseteq \mathcal{Q}$ as follows:

$$IND(\mathcal{P}) = \{(x, y) \in \mathcal{O} \times \mathcal{O} : \underset{a \in \mathcal{P}}{\forall} I_a(x) = I_a(y)\}$$

If a pair of objects belongs to $IND(\mathcal{P})$, the objects cannot be differentiated from each other given the attributes of the subset $\mathcal{P}$.

Before formally defining the construct concept, it is important to introduce the definitions of *discernibility relation* and *similarity relation* [13].

The *discernibility relation*, denoted as $DIS(\mathcal{P}) \subseteq \mathcal{O} \times \mathcal{O}$, is defined as follows:

$$DIS(\mathcal{P}) = \{(x, y) \in \mathcal{O} \times \mathcal{O} : (x, y) \notin IND(\mathcal{P})\}$$

This means that if a pair of objects (x, y) belongs to $DIS(\mathcal{P})$, then x and y differ from each other by at least one attribute from the set $\mathcal{P}$.

On the other hand, the *similarity relation*, denoted as $SIM(\mathcal{P}) \subseteq \mathcal{O} \times \mathcal{O}$, is defined for a set of attributes $\mathcal{P} \subseteq \mathcal{Q}$ as follows:

$$SIM(\mathcal{P}) = \{(x, y) \in \mathcal{O} \times \mathcal{O} : \underset{a \in \mathcal{P}}{\exists} I_a(x) = I_a(y)\}$$

If a pair of objects (x, y) belongs to $SIM(\mathcal{P})$, then these two objects are similar in at least one attribute from the set $\mathcal{P}$.

A construct (introduced by R. Susmaga in [13]) is an attribute subset of condition attributes that maintains the discernibility between pairs of objects belonging to different classes with the same capability as the whole set of condition attributes, while simultaneously ensuring similarity between objects belonging to the same class. The definition of a construct can be stated as follows:

Definition 1. *Given a DS, an attribute subset $\mathcal{R} \subseteq \mathcal{C}$ is a construct of the DS, if and only if $\mathcal{R}$ satisfies the following conditions:*

$$\underset{p \in \mathcal{O} \times \mathcal{O}}{\forall} \{[p \in DIS(\{d\}) \wedge p \in DIS(\mathcal{C})] \rightarrow p \in DIS(\mathcal{R})\} \tag{1}$$

$$\underset{p \in \mathcal{O} \times \mathcal{O}}{\forall} \{[p \in SIM(\{d\}) \wedge p \in SIM(\mathcal{C})] \rightarrow p \in SIM(\mathcal{R})\} \tag{2}$$

$$\underset{a \in \mathcal{R}}{\forall} \begin{cases} \underset{p \in \mathcal{O} \times \mathcal{O}}{\exists} p \in DIS(\{d\}) \wedge p \in DIS(\mathcal{R}) \wedge p \notin DIS(\mathcal{R} - \{a\}) \\ \qquad\qquad or \\ \underset{p \in \mathcal{O} \times \mathcal{O}}{\exists} p \in SIM(\{d\}) \wedge p \in SIM(\mathcal{R}) \wedge p \notin SIM(\mathcal{R} - \{a\}) \end{cases} \tag{3}$$

All attribute subsets satisfying conditions (1) and (2) are called super-constructs.

Condition (1) means that a construct discerns objects belonging to different classes. Moreover, according to condition (2), a construct guarantees similarity between objects belonging to the same class. Condition (3) ensures minimality regarding inclusion, which means that removing any attribute from $\mathcal{R}$ will make Conditions (1) or (2) invalid.

The set of all attributes contained in all constructs is called a *kernel*. It is important to highlight that the kernel could be empty.

Since the algorithm proposed in this article works on a binary matrix similar to the binary discernibility matrix, first, we will define the binary discernibility matrix, which is used by the most successful algorithms for computing all reducts.

The binary discernibility matrix *(BDM)*, proposed by Felix and Ushio [18], is a binary representation of the discernibility matrix proposed by Skowron and Rauszer [19].

Definition 2. *Given a DS, the BDM, is a matrix $\mathcal{M}$, where each row i results from comparing two objects (u and w) that do not belong to the same class $I_d(u) \neq I_d(w)$, i.e.,:*

$$\mathcal{M}_{i,j} = \{\delta_{c_j}(I_{c_j}(u), I_{c_j}(w))\}$$

where $I_{c_j}(u)$ denotes the value of the attribute $c_j \in \mathcal{C}$ in the object u, $I_d(u)$ denotes the value of the decision attribute in the object u, and δ_c is computed as follows:

$$\delta_c : V_c \times V_c \rightarrow \{1, 0\}$$

$$\delta_c(v_1, v_2) = \begin{cases} 0 & if\ v_1 = v_2 \\ 1 & otherwise \end{cases} \tag{4}$$

4. Proposed Algorithm

Our algorithm (*CC-GEAC*) goes through the subsets of attributes in a particular order, checking for each one if it is a construct. This verification is conducted on a binary matrix built from a decision system. This binary matrix was first introduced in [8]; it contains all pairs of object comparisons (attribute per attribute). In this work, we will refer to it as the *binary discernibility–similarity matrix (BDSM)*, and it is defined by:

$$\mathcal{M}_{i,j} = \begin{cases} \delta_{c_j}(I_{c_j}(u), I_{c_j}(w)) & if\ I_d(u) \neq I_d(w) \\ 1 - \delta_{c_j}(I_{c_j}(u), I_{c_j}(w)) & otherwise \end{cases}$$

where $I_{c_j}(u)$, $I_d(u)$, and δ_c are as in Definition 2.

The *BDSM* often contains redundant information, so it can be simplified. Next, some definitions used to simplify the *BDSM* will be introduced.

Definition 3. *Let $\mathcal{M}$ be a BDSM, i and f be two rows in $\mathcal{M}$, and i is a sub-row of f if and only if:*

(i) $\forall(column\ j\ in\ \mathcal{M})[(\mathcal{M}_{i,j} = 1) \Rightarrow (\mathcal{M}_{f,j} = 1)]$;
(ii) $\exists(column\ q\ in\ \mathcal{M})\ such\ that\ \mathcal{M}_{f,q} = 1\ and\ \mathcal{M}_{i,q} = 0$.

Example 1. *Suppose* $i = (001010)$ *and* $f = (101011)$ *are rows within a specific BDSM, i is a sub-row of f, since f has 1 everywhere i has 1, and there is at least one column, such that f has 1 and i has 0.*

Those rows for which there are no sub-rows are called basic rows and are defined as follows:

Definition 4. *Let* $\mathcal{M}$ *and f be defined as before. Row f is a basic row of* $\mathcal{M}$ *if and only if* $\nexists(row\ i\ in\ \mathcal{M})$ *such that i is sub-row of f.*

The matrix formed by the *basic rows* of the *BDSM*, without repeating, will be denoted as the *simplified binary discernibility–similarity matrix* (*SBDSM*).

CC-GEAC traverses the search space looking for super-constructs and then checks minimality regarding inclusion (Condition (3) of Definition 1); thus, it is necessary to define the concept of the super-construct in the *SBDSM*. Below, some definitions are provided to help define this concept.

Let $\mathcal{M}$ be an *SBDSM* and $\mathcal{A} \subseteq \mathcal{C}$ be an attribute subset. $\mathcal{M}^{\mathcal{A}}$ denotes the sub-matrix of $\mathcal{M}$, containing only columns corresponding to attributes in $\mathcal{A}$.

Definition 5. *Let* $\mathcal{M}$ *be an SBDSM,* $\mathcal{A} \subseteq \mathcal{C}$ *be an attribute subset, and r be a row of* $\mathcal{M}^{\mathcal{A}}$ *; r is denoted as a zero-row if it contains only zeros.*

Given an *SBDSM*, the super-construct definition (Definition 1) can be expressed as follows.

Proposition 1. *Let* $\mathcal{M}$ *and* $\mathcal{A}$ *as before.* $\mathcal{A}$ *is a super-construct in* $\mathcal{M}$ *if and only if in* $\mathcal{M}^{\mathcal{A}}$ *there is no zero-row.*

Proof. First, let us consider that $\mathcal{A}$ is a subset of attributes satisfying Proposition 1 and suppose that $\mathcal{A}$ is not a super-construct; hence, $\mathcal{A}$ does not satisfy Conditions (1) or (2) of Definition 1, then objects x and y exist in $\mathcal{O}$, $(x \neq y)$, such that if $I_d(x) = I_d(y)$, then $I_a(x) \neq I_a(y)$ for all a in $\mathcal{A}$, which, according to Definition 2, means that in $\mathcal{M}^{\mathcal{A}}$, there is a zero-row, or in another case, if $I_d(x) \neq I_d(y)$, then for all $a \in \mathcal{A}$, one has that $I_a(x) = I_a(y)$, and according to Definition 2, one has a zero-row in $\mathcal{M}^{\mathcal{A}}$. In either case, the hypothesis that $\mathcal{A}$ satisfies Proposition 1 is contradicted.

Let us assume that, as a hypothesis, $\mathcal{A}$ is a super-construct in $\mathcal{M}$, i.e., $\mathcal{A}$ satisfies Conditions (1) and (2) of Definition 1. And let us suppose that a zero-row exists in $\mathcal{M}^{\mathcal{A}}$, then according to Definition 2, there are either two objects of the same class $(I_d(x) = I_d(y))$ that differ in all attributes of $\mathcal{A}$, or there are two objects of different classes $(I_d(x) \neq I_d(y))$ that are indistinguishable in all attributes of $\mathcal{A}$, which contradicts the hypothesis; therefore, $\mathcal{A}$ satisfies the proposition 1. □

Proposition 1, analogous to Conditions (1) and (2) of Definition 1, ensures that a super-construct distinguishes between objects of different classes and guarantees similarity between objects of the same class. Therefore, if an attribute subset does not satisfy Proposition 1, this subset is not a construct.

CC-GEAC generates subsets of attributes following the lexicographic order, like in [20], to traverse the search space. Hereinafter, whenever we refer to a subset of condition attributes, it is considered to be an ordered set of attributes according to this lexicographical order. The $\oplus$ operator will be used to denote the concatenation of two disjoint ordered subsets of attributes. In this way, $\{c_1, c_3, c_5\} \oplus \{c_4, c_6, c_7\} = \{c_1, c_3, c_5, c_4, c_6, c_7\}$ and $\{c_4, c_6, c_7\} \oplus \{c_1, c_3, c_5\} = \{c_4, c_6, c_7, c_1, c_3, c_5\}$; note that by concatenating two ordered

subsets, the resulting subset maintains the order of the elements in these subsets, and the order in which these subsets are concatenated is relevant. Therefore, concatenation is non-commutative. Moreover, $\mathcal{P} \preceq \mathcal{Q}$ denotes that $\mathcal{P}$ precedes $\mathcal{Q}$ in the lexicographical order, and $\mathcal{P} \prec \mathcal{Q}$ denotes that $\mathcal{P}$ precedes $\mathcal{Q}$ but $\mathcal{P} \neq \mathcal{Q}$.

The *CC-GEAC* algorithm uses the gap concept introduced for the *LEX* algorithm [20]. The gap of an ordered attribute subset $\mathcal{A}$ is the rightmost attribute whose succeeding attribute in $\mathcal{A}$ is not its succeeding attribute in $\mathcal{C}$ (column in *SBDSM*); e.g., consider $\mathcal{C} = \{c_1, c_2, c_3, c_4, c_5\}$, for $\mathcal{A} = \{c_1, c_2, c_3\}$, there is no gap; for $\mathcal{A} = \{c_1, c_3, c_5\}$, the gap is c_3; finally, for $\mathcal{A} = \{c_1, c_2, c_4, c_5\}$, the gap is c_2.

The pruning property of gap elimination has been used for computing irreducible testors and reducts [10,20]. The same property can be enunciated, adapting it to the *BDSM* and the super-construct concept, as expressed in Proposition 2.

Proposition 2. *Let $\mathcal{M}$ be an SBDSM and $\mathcal{A} = \{c_{j_0}, \ldots, c_{j_s}\}$ be a construct, such that $\mathcal{A} \subset \mathcal{C}$ and c_{j_s} is the last attribute in $\mathcal{C}$. If there is a gap c_{j_p} in $\mathcal{A}$ and $\mathcal{A}' = \{c_{j_0}, \ldots, c_{j_{p-1}}, c_{j_{p+1}}\}$; then, there is no construct $\mathcal{P}$, such that $\mathcal{A} \prec \mathcal{P} \prec \mathcal{A}'$.*

Proof of Proposition 2. Let $\mathcal{W} = \{c_{j_0}, \ldots, c_{j_{p-1}}\}$, and $\mathcal{Z} = \{c_{j_{p+1}}, \ldots, c_{j_s}\}$, then $\mathcal{A}$ and $\mathcal{A}'$ can be written as $\mathcal{A} = \mathcal{W} \oplus \{c_{j_p}\} \oplus \mathcal{Z}$ and $\mathcal{A}' = \mathcal{W} \oplus \{c_{j_p+1}\}$. Let $\mathcal{P}$ be an ordered attribute subset, such that $\mathcal{A} \prec \mathcal{P} \prec \mathcal{A}'$, so $\mathcal{P}$ can be written as $\mathcal{P} = \mathcal{W} \oplus \{c_{j_p}\} \oplus \mathcal{V}$. Knowing that c_{j_p} is a gap (all attributes in $\mathcal{Z}$ are consecutive in $\mathcal{C}$) and c_{j_s} is the last attribute in $\mathcal{C}$, we can guarantee that $\mathcal{V} \subset \mathcal{Z}$ and, hence, $\mathcal{P} \subset \mathcal{A}$. From this, $\mathcal{P}$ cannot be a construct by condition (3) of Definition 1 (minimality condition). □

From Proposition 2, we have the following corollary.

Corollary 1. *Let $\mathcal{M}$ be as in Proposition 2, and let $\mathcal{A} = \{c_{j_0}, \ldots, c_{j_s}\}$ be a non-super-construct, such that c_{j_s} is the last attribute in $\mathcal{C}$. If there is a gap c_{j_p} in $\mathcal{A}$, and $\mathcal{A}' = \{c_{j_0}, \ldots, c_{j_{p-1}}, c_{j_{p+1}}\}$; then, there is no construct $\mathcal{P}$, such that $\mathcal{A} \prec \mathcal{P} \prec \mathcal{A}'$.*

Using Proposition 2 and Corollary 1, in both cases, all attribute subsets between $\mathcal{A}$ and $\mathcal{A}'$ can be discarded.

One of the recently proposed pruning strategies is also used in our proposal, which is based on the definition of attribute contribution, which, in our case, is adapted to the *SBDSM* as follows:

Definition 6. *Let $\mathcal{M}$ be as before, $\mathcal{A}$ be a subset of condition attributes, and c_i be a condition attribute, such that $c_i \notin \mathcal{A}$. The attribute c_i contributes to $\mathcal{A}$ if and only if the number of zero rows in the sub-matrix $\mathcal{M}^{\mathcal{A} \cup \{c_i\}}$ is less than the number of zero rows in the sub-matrix $\mathcal{M}^{\mathcal{A}}$.*

Definition 6 means that an attribute contributes to a given subset, which does not contain it if the new increased subset is capable of distinguishing more objects from different classes or making more objects from the same class appear similar than the original subset.

In *CC-GEAC*, the columns in *SBDSM* are represented as binary words, the length of which is equal to the number of rows in the matrix. The cumulative mask [12] for an attribute c_i, denoted as cm_{c_i}, is a binary representation of the ith column in the *SBDSM* and the cumulative mask for an attribute subset $\mathcal{A} = \{c_{j_0}, c_{j_1}, \ldots, c_{j_k}\}$ is defined as $cm_{\mathcal{A}} = cm_{c_{j_0}} \vee cm_{c_{j_1}} \vee \cdots \vee cm_{c_{j_k}}$ ($\vee$ denotes the binary *OR* operator). The number of $0's$ in $cm_{\mathcal{A}}$ is equal to the number of zero rows in $\mathcal{M}^{\mathcal{A}}$ for a given *SBDSM* $\mathcal{M}$. According to Definition 6, c_i contributes to $\mathcal{A}$ if and only if $cm_{\mathcal{A} \oplus \{c_i\}}$ has more *1's* than $cm_{\mathcal{A}}$. The proposed algorithm generates subsets of attributes in increasing order following the lexicographical order introduced in [20], so the cumulative mask can be computed as $cm_{\mathcal{A} \oplus \{c_i\}} = cm_{\mathcal{A}} \vee cm_{c_i}$. Notice that, from Definition 6, c_i contributes to $\mathcal{A}$ if and only if

$cm_{\mathcal{A}\oplus\{c_i\}} \neq cm_{\mathcal{A}}$ since $cm_{\mathcal{A}\oplus\{c_i\}}$ cannot have less 1's than $cm_{\mathcal{A}}$. From Proposition 1, it can be seen that $\mathcal{A} \subseteq \mathcal{C}$ is a super-construct if and only if $cm_{\mathcal{A}} = (1, \ldots, 1)$.

The exclusion mask, defined for the first time in [21], is relevant to determine whether or not a super-construct is a construct.

Definition 7. *Let $\mathcal{M}$ and $\mathcal{A}$ be as before. $em_{\mathcal{A}}$ denotes the exclusion mask of $\mathcal{A}$, and it is defined as the binary word that has a 1 in the ith bit if the ith row in $\mathcal{M}$ has a 1 in only one of the columns corresponding to the attributes in $\mathcal{A}$; the remaining bits of $em_{\mathcal{A}}$ are zeros. Let c_i be an attribute, such that $c_i \notin \mathcal{A}$. The exclusion mask of $\mathcal{A} \oplus \{c_i\}$ is computed as follows:*

$$em_{B\oplus\{c_i\}} = (em_B \wedge \neg cm_{c_i}) \vee (\neg cm_B \wedge cm_{c_i}) \tag{5}$$

where cm is the cumulative mask, and $\neg$ is the binary operator for negation.

Example 2. *Let $\mathcal{A}_0 = \{c_0, c_1\}$, $\mathcal{A}_1 = \{c_0, c_2, c_3\}$, and $\mathcal{A}_2 = \{c_4, c_5\}$ be subsets of attributes of a SBDSM:*

	c_0	c_1	c_2	c_3	c_4	c_5
r_0	1	0	1	0	1	0
r_1	0	1	0	1	1	1
r_2	1	1	0	1	0	0
r_3	1	0	0	1	1	1
r_4	0	0	1	1	1	0
r_5	0	1	1	1	0	1
r_6	1	0	1	1	0	0

The corresponding cumulative and exclusion masks are as follows:

$$cm_{\mathcal{A}_0} = (1,1,1,1,0,1,1), em_{\mathcal{A}_0} = (1,1,0,1,0,1,1),$$

$$cm_{\mathcal{A}_1} = (1,1,1,1,1,1,1), em_{\mathcal{A}_1} = (0,1,0,0,0,0,0),$$

$$cm_{\mathcal{A}_2} = (1,1,0,1,1,1,0), em_{\mathcal{A}_2} = (1,0,0,0,1,1,0)$$

The following proposition represents a contribution to the cumulative calculation of the exclusion mask.

Proposition 3. *Let $\mathcal{M}$ and $\mathcal{A}$ be as defined in Definition 7. Let $c_i \notin \mathcal{A}$. If $\exists c_x \in \mathcal{A}$, such that $em_{\mathcal{A}\oplus\{c_i\}} \wedge cm_{c_x} = (0, \ldots, 0)$, then $\mathcal{A} \oplus \{c_i\}$ cannot be a subset of any construct, and c_i is said to be exclusionary with $\mathcal{A}$.*

Proof of Proposition 3. First, suppose that $\exists c_x \in \mathcal{A}$, such that $em_{\mathcal{A}\oplus\{c_i\}} \wedge cm_{c_x} = (0, \ldots, 0)$. Now, suppose that $\mathcal{A} \oplus \{c_i\}$ is a subset of a construct; thus, c_x is an essential attribute, which implies that there is a zero-row r_x in the sub-matrix $\mathcal{M}^{\mathcal{A}\oplus\{c_i\}-\{c_x\}}$, such that the bit in cm_{c_x} associated with the row r_x is 1. Thus, by Definition 7, the bit in $em_{\mathcal{A}\oplus\{c_i\}}$ is associated with the row r_x is 1. So we can conclude that $em_{\mathcal{A}\oplus\{c_i\}} \wedge cm_{c_x} \neq (0, \ldots, 0)$ if $\mathcal{A} \oplus \{c_i\}$ is a subset of a construct, which is a contradiction $\perp$. □

Proposition 3 allows all supersets of $\mathcal{A} \oplus \{c_i\}$ to be discarded when c_i is exclusionary with $\mathcal{A}$. In what follows, the application of Proposition 3 will be referred to as the exclusion evaluation.

Proposition 4. *Let $\mathcal{A}$ be a subset of condition attributes and c_i be a condition attribute, such that $c_i \notin \mathcal{A}$. $\mathcal{A} \oplus \{c_i\}$ is a construct if and only if $cm_{\mathcal{A}\oplus\{c_i\}} = (1, \ldots, 1)$ and c_i is non-exclusionary with $\mathcal{A}$.*

Proof of Proposition 4. First, suppose that $cm_{\mathcal{A}\oplus\{c_i\}} = (1,\ldots,1)$; therefore, $\mathcal{A}\oplus\{c_i\}$ is a super-construct by Proposition 1. Then, if c_i is non-exclusionary with $\mathcal{A}$, all attributes in $\mathcal{A}\oplus\{c_i\}$ are essential attributes by Proposition 3, which means that removing an attribute from $\mathcal{A}\oplus\{c_i\}$ would invalidate any (or both) of the Conditions (1) or (2) of Definition 1. Thus, we can conclude that $\mathcal{A}\oplus\{c_i\}$ is a construct. □

To reduce the search space, Proposition 5 is used to determine a priori if a zero-row prevails after adding any of the remaining attributes to the current attribute subset. This idea was introduced in [22] as a pruning strategy to compute the minimum-length irreducible testor. Similarly, it can be used to prune the search space for computing constructs.

Proposition 5. *Let $\mathcal{M}$ be an SBDSM, $\mathcal{A} = \{c_{j_0}, c_{j_1}, \ldots, c_{j_p}\}$ be a subset of condition attributes, and c_i be a condition attribute, such that $c_i \notin \mathcal{A}$. If there is a zero-row r in $\mathcal{M}^{\mathcal{A}}$, such that $\mathcal{M}_{r,c_i} = \mathcal{M}_{r,c_{i+1}} = \cdots = \mathcal{M}_{r,c_n} = 0$, then $\mathcal{A}\oplus\{c_i\}$ is not contained in any construct.*

Proof of Proposition 5. Since there is a zero-row r in $\mathcal{M}^{\mathcal{A}}$, $\mathcal{A}$ is not a construct. By hypothesis, if we consider $\mathcal{M}^{\mathcal{A}\oplus\{c_i\}}$, the row r prevails as a zero-row, even if we add any attribute in $\{c_{i+1},\ldots,c_n\}$. So, we can conclude that $\mathcal{A}\oplus\{c_i\}$ will not be a subset of any construct. □

Corollary 2. *Let $\mathcal{M}$ be an SBDSM, $\mathcal{A} = \{c_{j_0}, c_{j_1}, \ldots, c_{j_p}\}$ be a non-super-construct, and c_i be a condition attribute, such that $c_i \notin \mathcal{A}$. If there is a zero-row r in $\mathcal{M}^{\mathcal{A}}$, such that $\mathcal{M}_{r,c_i} = \mathcal{M}_{r,c_{i+1}} = \cdots = \mathcal{M}_{r,c_n} = 0$, then there is no construct $\mathcal{P}$, such that $\mathcal{A} \prec \mathcal{P} \prec \mathcal{A}'$, where $\mathcal{A}' = \{c_{j_0}, \ldots, c_{j_p+1}\}$.*

Consequently, following Proposition 5 and Corollary 2, all attribute subsets between $\mathcal{A}$ and $\mathcal{A}'$ can be discarded.

Next, we introduce the definition of a zero mask to determine, a priori and in constant time, whether a zero-row will prevail after adding any remaining attributes to the current attribute subset.

Definition 8. *Let $\mathcal{M}$ be as before; we define the zero mask of a condition attribute c_i, denoted as zm_{c_i}, to the binary word that has a one in the ith bit if the ith row in $\mathcal{M}$ has a one in any of the (at least one) columns corresponding to the condition attributes c_j, such that $j \geq i$, and it is 0 otherwise. The zero mask of c_i is computed as follows:*

$$zm_{c_i} = cm_{c_i} \vee cm_{c_{i+1}} \vee cm_{c_{i+2}} \vee \cdots \vee cm_{c_n}$$

where cm refers to the cumulative mask.

By using Definition 8, we can discard some subsets of attributes, thereby pruning the search space. To facilitate this, the following proposition is introduced.

Proposition 6. *Let $\mathcal{M}$, $\mathcal{A}$, and c_i be as in Proposition 5. If $zm_{c_i} \vee cm_{\mathcal{A}} \neq (1,\ldots,1)$, then $\mathcal{A}\oplus\{c_i\}$ is not a subset of any construct.*

Proof of Definition 8. Since $zm_{c_i} \vee cm_{\mathcal{A}} \neq (1,\ldots,1)$, there is a zero-row r in $\mathcal{M}^{\mathcal{A}}$ that remains as a zero-row in $\mathcal{M}^{\mathcal{A}\oplus\{c_i\}}$, even if we add any attribute in $\{c_{i+1},\ldots,c_n\}$. So, we can conclude that $\mathcal{A}\oplus\{c_i\}$ is not a subset of any construct. □

CC-GEAC, unlike the algorithms reported in [10–12], uses the kernel [7] and the minimum attribute subset [11] together to reduce the number of generated subsets.

Next, the kernel definition is introduced, given a simplified binary discernibility–similarity matrix.

Definition 9. *Let $\mathcal{M}$ be an SBDSM and $\mathcal{C}$ be the set of condition attributes associated with the columns in $\mathcal{M}$. The kernel of $\mathcal{M}$ is defined as:*

$$Kernel(\mathcal{M}) = \{c_q \in \mathcal{C} \mid \exists f : \mathcal{M}_{f,q} = 1 \text{ and } \forall (i \neq q) : \mathcal{M}_{f,i} = 0 \mathcal{M}_{f,q} = 1\}$$

Given a simplified binary discernibility–similarity matrix (*SBDSM*), according to Definition 9, if, in *SBDSM*, there exists a row with all zeros except for one column, the attribute corresponding to the column with the value 1 is indispensable for discriminating objects from different classes or maintaining similarity between objects in the same class. Therefore, this attribute must belong to every construct and, consequently, to the kernel. Attributes that meet this condition, and only those, belong to the kernel. It is evident that the kernel is a subset of every construct.

On the other hand, following [11], those attributes in the row with the least amount of 1's belong to the minimum attribute subset *(MSA)*. Therefore, if the kernel is not empty, *MSA* has one attribute, which also belongs to the kernel. For this reason, in this paper, the rows with only a "1" are not considered to compute the *MSA*. Accordingly, given an *SBDSM*, the definition of the minimum attribute subset *(MSA)* used in this paper is as follows:

Definition 10. *Let $\mathcal{M}'$ be the sub-matrix resulting from eliminating the rows with just a 1 in the SBDSM, $\mathcal{C}$ be the set of condition attributes, and f be the row with the least number of 1's in $\mathcal{M}'$ (if there is more than one row, any one of them can be selected). The minimum attribute subset (MSA) is the subset:*

$$MSA(\mathcal{M}') = \{c_q \in \mathcal{C} \mid \mathcal{M}'_{f,q} = 1\}$$

In *CC-GEAC*, the kernel is computed first. Then if the kernel is a construct, it will be the unique construct in the *SBDSM*. Otherwise, if the kernel is not a construct, to search for constructs, only those subsets contained in $\mathcal{C}'$ will be taken into account by the algorithm, being:

$$\mathcal{C}' = MSA(\mathcal{M}) \oplus (\mathcal{C} - (MSA(\mathcal{M}) \oplus Kernel(\mathcal{M}))) \tag{6}$$

The attributes in $\mathcal{C}' = \{c'_0, c'_1, \ldots, c'_q, \ldots, c'_p\}$ are relabeled, such that the first q attributes belong to the *MSA* following the order in $\mathcal{C}$, and the remaining $p - q$ attributes are attributes in $\mathcal{C}$ that neither belong to the kernel nor the *MSA*; the last $p - q$ attributes also follow the order in $\mathcal{C}$, e.g., let $\mathcal{C} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$, $MSA = \{c_3, c_5\}$, and the $Kernel = \{c_2\}$, then according to Equation (6) $\mathcal{C}' = \{c'_0, c'_1, c'_2, c'_3, c'_4\}$, where $c'_0 = c_3$, $c'_1 = c_5$, $c'_2 = c_1$, $c'_3 = c_4$, $c'_4 = c_6$.

CC-GEAC uses Proposition 5 and Corollary 2 to determine if there is a zero-row after adding all the remaining attributes in $\mathcal{C}'$ to the current attribute subset. If there is no zero-row, the attribute contribution condition is verified. If the new attribute contributes, it is checked whether the current candidate (the current attribute subset along with the added attribute) is a super-construct. If the current candidate is a super-construct, then, using Proposition 3, *CC-GEAC* determines whether the candidate is a construct; if it is a construct, the candidate is saved. At this stage, the algorithm generates the following attribute subset, pruning those subsets that cannot lead to a construct based on Corollary 2 and Corollary 1. The *CC-GEAC* algorithm continues this process until the first attribute in the current attribute subset no longer belongs to the *MSA*.

Algorithm 1 shows the pseudo-code of *CC-GEAC*. The *CC-GEAC* algorithm operates over the simplified binary discernibility–similarity matrix *(SBDSM)*. First, the *kernel* is computed. If the kernel is a super-construct, the algorithm ends by returning the kernel as the only construct. Otherwise, the *MSA* is computed to form set $\mathcal{C}'$, which will be considered to generate the candidate subsets to evaluate. The current attribute subset is initialized with the first attribute in $\mathcal{C}'$, and then the cumulative mask is updated. Then, if after adding one of the remaining attributes to the current attribute subset, there is no zero-

row (using Proposition 6), the attribute contribution (Definition 6) is evaluated. For those attribute subsets with a contributing attribute (candidates), the super-construct condition (Proposition 1) is evaluated. Proposition 3 is applied to each found super-construct to determine whether it is a construct that will be saved in the result (SC). Candidate evaluation ends at this point, and the next attribute subset in lexicographical order is generated by calling *nextSet()*.

Algorithm 1: All Constructs Computation

Input : $\mathcal{M}$, an *SBDSM*; $\mathcal{C}$, the set of condition attributes
Output: SC, set of all constructs

$SC \leftarrow \emptyset$
$\mathcal{A} \leftarrow \emptyset$ // Attribute subset (candidate).
$\mathcal{K} \leftarrow Kernel(\mathcal{M})$
if $cm_{\mathcal{K}} = (1, \ldots, 1)$ **then**
  // The kernel is the only construct in the *SBDSM*
  $SC \leftarrow \mathcal{K}$
  return SC
end
$SA \leftarrow MSA(\mathcal{M})$
$\mathcal{C}' \leftarrow SA \oplus (\mathcal{C} - (SA \oplus \mathcal{K}))$ // $\mathcal{C}' = \{c'_0, c'_1, \ldots, c'_p\}$, where p is the index of the last attribute in $\mathcal{C}'$
$next \leftarrow True$
$i \leftarrow 0$ // Index of the new attribute in $\mathcal{C}'$ to add in the current attribute subset, $i = (0, 1, \ldots, p)$.
while *next* **do**
  $construct, superConstruct, contributes, zeroRows \leftarrow False$
  $cm_{\mathcal{A} \oplus \{c'_i\}} \leftarrow updateCM(\mathcal{A}, i)$
  if $zm_{c'_i} \vee cm_{\mathcal{A}} \neq (1, \ldots, 1)$ **then**
    $zeroRows \leftarrow True$
  end
  else
    if $cm_{\mathcal{A} \oplus \{c'_i\}} \neq cm_{\mathcal{A}}$ **then**
      $contributes \leftarrow True$
      if $cm_{\mathcal{A} \oplus \{c'_i\}} = (1, \ldots, 1)$ **then**
        $superConstruct \leftarrow True$
        $construct \leftarrow exclusion(\mathcal{A}, i)$
        if *construct* **then**
          $SC \leftarrow SC \oplus \{\mathcal{K} \oplus \mathcal{A} \oplus \{c'_i\}\}$
        end
      end
    end
  end
  $next, \mathcal{A}, i \leftarrow nextSet(\mathcal{A}, i, contributes, superConstruct, construct, zeroRows)$
end
return SC

The procedure for updating the *CM* of a candidate $(\mathcal{A} \oplus \{c'_i\})$ appears in Algorithm 2. The *CM* is stored in an array, indexed by the index in $\mathcal{C}'$ of the last attribute in the current attribute subset $(\mathcal{A})$. The last attribute in $\mathcal{B}$ is determined using $last(\mathcal{B})$, while $cm_{\mathcal{K}}$ stores the *CM* of the kernel.

The gap elimination is performed through Algorithm 3. Every consecutive attribute in the current attribute subset, beginning from the last attribute, is removed. In Algorithm 3, the variable q keeps the index in $\mathcal{C}'$ of the last attribute in $\mathcal{A}$, while p keeps track of the index of the last attribute in $\mathcal{C}'$.

Algorithm 4 performs the exclusion evaluation. Firstly, it computes the exclusion mask by applying Equation (5) (see Definition 7). Then by applying Proposition 3, each attribute in the current attribute subset $\mathcal{A}$ is evaluated for exclusion. Thus, using Proposition 4, it is possible to determine whether or not the current candidate $\mathcal{A} \oplus \{c'_i\}$ is a construct.

Algorithm 2: UpdateCM

Input : $\mathcal{A}, i$
Output: $cm_{\mathcal{A}\oplus\{c'_i\}}$

if $\mathcal{A} = \emptyset$ **then**
  $cm_{\mathcal{A}} \leftarrow cm_{\mathcal{K}}$ // All attribute subsets must contain the kernel.
end
else
  $c'_q = last(\mathcal{A})$
  $cm_{\mathcal{A}} \leftarrow mask[q]$
end
$cm_{\mathcal{A}\oplus\{c'_i\}} \leftarrow (cm_{\mathcal{A}} \vee cm_{c'_i})$
$c'_q = last(\mathcal{A})$
$mask[q] \leftarrow cm_{\mathcal{A}\oplus\{c'_i\}}$
return $cm_{\mathcal{A}\oplus\{c'_i\}}$ // Updated cumulative mask (*CM*)

Algorithm 3: RemoveGAP

Input : $\mathcal{A}$
Output: $\mathcal{A}$

$c'_p \leftarrow last(\mathcal{C}')$ // The last attribute in $\mathcal{C}'$.
while $last(\mathcal{A}) = (p-1)$ **do**
  $c'_q \leftarrow last(\mathcal{A})$
  $p \leftarrow q$
  $\mathcal{A} \leftarrow \mathcal{A} - \{c'_q\}$
  if $|\mathcal{A}| = 1$ **then**
    break
  end
end
return $\mathcal{A}$

Algorithm 4: Exclusion

Input : $\mathcal{A}, i$
Output: *construct*

$em, cm \leftarrow (0, \ldots, 0)$
foreach $x \in \mathcal{A} \oplus \{c'_i\}$ **do**
  $em \leftarrow (em \wedge \neg cm_x) \vee (\neg cm \wedge cm_x)$
  $cm \leftarrow cm \vee cm_x$
end
$construct \leftarrow True$
foreach $x \in \mathcal{A}$ **do**
  if $em \vee cm_x = (0, \ldots, 0)$ **then**
    $construct \leftarrow False$
    break
  end
end
return *construct*

Algorithm 5 shows the pseudocode for generating the following attribute subset $\mathcal{A}$ and determining the index in $\mathcal{C}'$ of the next attribute to be processed according to the lexicographic order. First, using Proposition 6, it is determined if there is a zero-row after adding to $\mathcal{A}$ all of the remaining c'_q attributes, for $q \geq i$, where i is the index in $\mathcal{C}'$ of the current attribute. If there is no zero-row, if the last attribute in $\mathcal{C}'$ is included in the current candidate, and if the current candidate is not a super-construct (Proposition 2) or a construct (Proposition 4), the gap is removed. Otherwise, the following attribute subset is generated in lexicographical order as follows:

1. If the current candidate is a super-construct or the current attribute c'_i does not contribute to $\mathcal{A}$; the next attribute in $\mathcal{C}'$ is then considered, which is c'_{i+1}. Thus, if the current candidate is a super-construct or if c'_i does not contribute to $\mathcal{A}$, all supersets of $\mathcal{A} \oplus \{c'_i\}$ are discarded.

2. If the current candidate $(\mathcal{A} \oplus \{c'_i\})$ is not a super-construct and c'_i contributes to $\mathcal{A}$; then c'_i is included in $\mathcal{A}$ and the next attribute in $\mathcal{C}'$ is set as the current attribute, i.e., $i \leftarrow i + 1$.

Algorithm 5: NextSet

```
Input  : A, i, contributes, superConstruct, construct, zeroRows
Output: next, A, i
next ← True
if zeroRows then
    c'_q ← last(A)
    A ← A − {c'_q}
    i ← q + 1
end
else
    if i = p then
        if construct or not superConstruct then
            removeGAP(A)
        end
        c'_q ← last(A)
        i ← q + 1
        A ← A − {c'_q}
    end
    else
        if not contributes or superConstruct then
            i ← i + 1
        end
        else
            A ← A ⊕ {c'_i}
            i ← i + 1
        end
    end
end
if A = ∅ and i = (|SA| − 1) then
    next ← False
end
return next, A, i
```

As described above, the search space is traversed following the lexicographical order, evaluating some attribute subsets and discarding others. First, the kernel and the *MSA* are computed to reduce the number of generated and evaluated attribute subsets (the kernel is contained in all constructs, and every construct must have at least one attribute from the *MSA*). The time complexity in the worst case for computing the kernel and the *MSA* is $\Theta(nm)$, where n is the number of columns and m is the number of rows in the *SBDSM*. On the other hand, the time complexity for updating the cumulative mask, checking for zero rows, and determining if the current attribute subset is a super-construct is $\Theta(m)$ in all cases. To check if a super-construct is a construct, firstly, the exclusion mask is computed; after that, the attribute exclusion is evaluated, which is $\Theta(nm)$. Generating the following attribute subset, we have $\Theta(n)$, as in the worst-case scenario, all n attributes of the current subset must be analyzed. Since the exclusion evaluation has the highest time complexity in *CC-GEAC*, the time complexity for evaluating a candidate is $\Theta(nm)$. Therefore, since, in the worst case, the number of evaluated candidates has an exponential relation with the number of attributes (n), *CC-GEAC* has an exponential complexity in terms of the number of attributes. Thus, our algorithm has the same complexity as the algorithms designed to compute reducts or *IT* that can be used for computing constructs following the process described in [8]. In the next section, some experiments regarding runtimes showing the performance of *CC-GEAC* are discussed.

5. Experiments and Results

The *CC-GEAC* algorithm uses the binary discernibility–similarity matrix to compute constructs following the process described in [8]. Hence, to assess our algorithm in Section 5.1, we present a runtime comparison of *CC-GEAC* against the option of computing constructs following [8] using the fastest algorithms for reducts computation currently available: Fast-CT-EXT [11], FastBR [12], and GCreduct [10], over *SBDSMs* computed from real-world datasets taken from [23–25]. Subsequently, in Section 5.2, we present a second experiment using artificial *SBDSMs*. We used the authors' implementation of Fast-CT-EXT, FastBR, and GCreduct for our experiments, jointly with *CC-GEAC*, all implemented in Java. To carry out our experiments, a computer with two Intel Xeon processors at 2.40 GHz, with 256 GB of RAM, and with a Windows 10 64-bit operating system was used.

5.1. Experiments on Real-World Datasets

A total of 34 real-world datasets were selected from the UCI Machine Learning Repository [23], Keel Repository [24], and OpenML Repository [25]. These datasets have *SBDSMs* with different densities, which allows representing the runtime variations among the four algorithms with respect to the variation of the *SBDSM* density. The density is computed by dividing the number of ones in the matrix by the total number of cells. Additionally, the discretization method described in [26] was employed for numerical attributes using Weka.

Table 1 shows the 34 datasets used in our experiment. The *Dataset* column specifies the dataset's name. The columns *Instances*, *Attributes*, and *Classes* show the number of objects, attributes, and classes of the respective dataset. The *Size* column indicates the size of the *SBDSM*, based on the number of rows and columns. The *Density* column shows the density of the *SBDSM*. The *Constructs* column shows the number of constructs. On the other hand, Table 2 shows the number of evaluated candidates and runtimes (in seconds) for each dataset in Table 1, as computed by the four algorithms. The cells containing the shortest runtime or the smallest number of assessed candidates for each dataset are highlighted in bold red.

According to Table 2, *CC-GEAC* shows the best performance in *SBDSM* with a density under 0.34. From Table 2, it can be seen that *CC-GEAC* is outperformed in some cases for *SBDSM* with a density under 0.34. However, in those cases, the difference in runtimes is quite small, less than 16 milliseconds. From Table 2, it can also be noted that for *SBDSM* with a density under 0.34, the number of candidates evaluated by *CC-GEAC* in all cases is the smallest. On the other hand, Fast-BR outperformed *CC-GEAC* for most simplified binary discernibility–similarity *SBDSM* with a density higher than 0.34. Indeed, as can be seen in Table 2, as the density increases, *CC-GEAC* loses efficiency since fewer subsets are eliminated and the evaluation cost of each subset in our algorithm is higher compared to Fast-BR.

This experiment concludes that *CC-GEAC* is the best option for computing constructs for *SBDSM* with a density under 0.34, while Fast-BR is the best option for *SBDSM* with a density higher than 0.34.

Table 1. Datasets used in the experiments and some of their characteristics.

Dataset	Instances	Attributes	Classes	Size	Density	Constructs
Landsat_tst [23]	2000	36	6	36×36	0.028	1
Kr-vs-kp [23]	3196	36	2	29×36	0.029	4
Ionosphere [24]	351	33	2	30×33	0.031	2
Hypothyroid [25]	3772	29	4	25×29	0.036	15,932
Spect-Heart [23]	267	44	2	39×44	0.037	36
Connect-4 [23]	676	42	3	241×42	0.047	84
Wdbc [25]	569	30	2	18×30	0.056	29
Vehicle [25]	846	18	4	17×18	0.056	1
Primary-tumor [24]	339	17	21	16×17	0.059	1

Table 1. *Cont.*

Dataset	Instances	Attributes	Classes	Size	Density	Constructs
LtterUV [25]	1577	16	2	16 × 16	0.063	1
Vote [23]	435	16	2	13 × 16	0.067	2
Segment [25]	2310	19	7	9 × 19	0.070	4
Credit-Approval [25]	690	15	2	14 × 15	0.071	2
Heart-c [23]	303	13	5	12 × 13	0.083	2
Led24 [24]	3200	24	10	75 × 24	0.103	47
Flags [23]	194	28	4	36 × 28	0.106	177
German-credit [25]	1000	20	2	30 × 20	0.110	23
Heart-h [23]	294	13	5	14 × 13	0.165	8
Lsd [23]	266	35	15	119 × 35	0.233	6536
Zoo [23]	101	17	7	21 × 17	0.235	46
Student-Por [23]	649	32	20	2588 × 32	0.242	135,743
Dermatology [24]	366	34	6	522 × 34	0.262	82,228
Lymph [23]	148	18	4	193 × 18	0.265	827
QSAR-Biodeg [23]	1055	41	2	278 × 41	0.298	28,941
Mushroom [23]	8124	21	2	56 × 21	0.299	1038
Student-Mat [23]	395	32	20	3755 × 32	0.305	487,613
Hepatitis [23]	155	19	2	201 × 19	0.342	1754
Sponge [23]	76	45	2	99 × 45	0.346	24,209
Labor [24]	57	16	2	90 × 16	0.351	458
Diabetes130US [25]	1018	35	3	3330 × 35	0.374	1,046,864
Cylinder-Bands [23]	540	39	2	5771 × 39	0.390	1,850,264
Sick [24]	3772	29	2	830 × 29	0.480	15,372
Lung-Cancer [23]	32	56	3	446 × 56	0.492	20,669,421
Divorce [24]	170	54	2	2941 × 54	0.565	51,736,610

Table 2. Number of evaluated candidates and runtimes (in seconds) spent by Fast-CT-EXT, FastBR, GCreduct, and *CC-GEAC*.

Name	Fast-CT_EXT		Fast-BR		GCreduct		CC-GEAC	
	Candidates	Time	Candidates	Time	Candidates	Time	Candidates	Time
Landsat_tst	3.44×10^{10}	549.268	630	0.016	36	**<0.001**	**1**	0.016
Kr-vs-kp	6.23×10^{8}	9.773	3683	0.016	29	**<0.001**	**5**	0.004
Ionosphere	8.06×10^{8}	11.542	4090	0.026	30	**<0.001**	**2**	0.004
Hypothyroid	5.24×10^{7}	0.602	2325	0.015	25	**<0.001**	**2**	0.003
Spect-Heart	3.27×10^{10}	618.079	7.44×10^{9}	902.853	1.55×10^{10}	335.687	**2047**	**<0.001**
Connect-4	4.33×10^{9}	140.182	7.90×10^{8}	145.276	2.53×10^{8}	9.218	**37,122**	**0.063**
Wdbc	1.38×10^{6}	0.049	1.94×10^{5}	0.052	1.92×10^{5}	0.014	**101**	0.003
Vehicle	1.31×10^{5}	0.011	153	0.003	18	**<0.001**	**1**	**<0.001**
Primary-tumor	32,784	0.016	576	0.015	16	**<0.001**	**1**	**<0.001**
LtterUV	32,784	**<0.001**	120	**<0.001**	16	**<0.001**	**1**	**<0.001**
Vote	9312	0.001	299	0.002	13	**<0.001**	**3**	0.003
Segment	1596	**<0.001**	93	0.016	9	**<0.001**	**4**	0.003
Credit-Approval	12,296	**<0.001**	91	**<0.001**	14	**<0.001**	**2**	0.003
Heart-c	3104	0.015	66	0.015	12	**<0.001**	**2**	0.002
Led24	8.11×10^{6}	0.172	2.32×10^{6}	0.312	2.84×10^{5}	0.031	**572**	**0.015**
Flags	2.28×10^{7}	0.300	4.18×10^{6}	0.288	2.87×10^{6}	0.064	**16,105**	**0.010**
German-credit	3.92×10^{5}	0.063	1.79×10^{5}	0.058	24,098	0.018	**336**	**<0.001**
Heart-h	1974	**<0.001**	740	**<0.001**	616	**<0.001**	**48**	**<0.001**
Lsd	1.48×10^{8}	3.589	1.47×10^{7}	1.531	1.46×10^{8}	3.938	**4.81×10^{5}**	**0.156**
Student-Por	3.04×10^{9}	292.027	1.31×10^{9}	239.086	2.01×10^{8}	27.968	**7.41×10^{7}**	**23.748**
Dermatology	7.66×10^{8}	25.323	8.63×10^{7}	8.346	6.65×10^{8}	26.413	**5.10×10^{7}**	**4.756**

Table 2. *Cont.*

Name	Fast-CT_EXT		Fast-BR		GCreduct		CC-GEAC	
	Candidates	Time	Candidates	Time	Candidates	Time	Candidates	Time
Zoo	14,131	**0.002**	3953	0.005	13252	**0.002**	**1220**	0.003
Lymph	179,763	**0.015**	100,255	0.031	72,575	0.031	**50,351**	0.063
QSAR-Biodeg	5.53×10^{10}	1853.365	1.25×10^{9}	169.121	3.65×10^{10}	1242.256	$\mathbf{1.96 \times 10^{7}}$	**1.526**
Mushroom	150,248	**0.016**	36,220	0.031	105,139	**0.016**	**38,573**	**0.016**
Student-Mat	2.38×10^{9}	342.415	8.34×10^{8}	223.088	5.49×10^{8}	116.848	$\mathbf{1.90 \times 10^{8}}$	**79.715**
Hepatitis	286,082	0.063	1.39×10^{5}	0.083	1.38×10^{5}	**0.047**	**88,004**	0.094
Sponge	4.15×10^{8}	7.251	$\mathbf{2.91 \times 10^{6}}$	**0.313**	2.44×10^{8}	4.500	2.92×10^{7}	1.552
Labor	31,105	**<0.001**	16,535	0.015	22,275	**<0.001**	**11,043**	0.009
Diabetes130US	1.54×10^{9}	574.130	$\mathbf{2.41 \times 10^{8}}$	**119.906**	1.13×10^{9}	480.050	3.39×10^{8}	210.349
Cylinder-Bands	8.73×10^{9}	4141.789	$\mathbf{6.27 \times 10^{8}}$	**338.837**	5.04×10^{9}	2812.581	2.65×10^{9}	946.929
Sick	4.75×10^{5}	0.077	**21,350**	0.031	46,990	0.018	55,147	**0.038**
Lung-Cancer	2.85×10^{10}	2784.221	$\mathbf{5.61 \times 10^{8}}$	**85.886**	2.05×10^{10}	2118.727	1.19×10^{10}	1075.469
Divorce	6.27×10^{10}	12,609.659	$\mathbf{4.64 \times 10^{9}}$	**1065.11**	5.16×10^{10}	9034.070	2.28×10^{10}	7029.546

The cells containing the shortest runtime or the smallest number of assessed candidates for each dataset are highlighted in bold red.

5.2. Experiments on Artificial SBDSMs

Another experiment was performed with the aim of further evaluating the performance of *CC-GEAC*, with respect to the density of the SBDSMs. In this experiment, the runtime and the number of evaluated candidates of the algorithm are compared against those obtained by the same aforementioned algorithms. This experiment was performed over artificial *SBDSMs* generated in an ample interval of densities; 900 artificial SBDSMs were generated with 30 columns and between 30 and 1000 rows. These dimensions were selected to guarantee runtimes within certain reasonable limits and generate different densities. The densities of the matrices used in our experiments range from 0.01 to 0.9, which allows for a demonstration of the performance of the algorithms as the density of the matrices varies. Figures 1 and 2 show, on a logarithmic scale, the average runtime and the average number of evaluated candidates for all *SBDSMs* with the "same density" (taking into account two decimal points), sorted in acceding order by their density. From these figures, it can be seen that *CC-GEAC* is the fastest for all *SBDSMs* with a density lower than 0.29.

Three distinct Wilcoxon right-tailed tests were performed using 300 matrices with densities less than 0.29 from the 900 artificial *SBDSMs* containing 30 columns. Each test had a 95% confidence level and an alpha value set at 0.05. The objective was to evaluate the performance of *CC-GEAC* in computing constructs compared to the Fast-CT_EXT, Fast-BR, and GCreduct algorithms, when each is used for the same purpose with the *SBDSM*. The results of these tests show a p-value of 0.00 for each test. These results confirm that *CC-GEAC* is statistically faster than Fast-CT_EXT, Fast-BR, and GCreduct algorithms for computing constructs using *SBDSM* with a density of less than 0.29.

Furthermore, to delve into the performance of *CC-GEAC* for *SBDSMs* with a density of less than 0.29, 300 artificial *SBDSMs* were generated with 40 columns, and between 40 and 1000 rows. This also allows us to show the behavior of *CC-GEAC* as long as the number of columns in the *SBDSMs* increases. Figures 3 and 4 show the average results in the logarithmic scale regarding the runtime and the number of evaluated candidates over artificial *SBDSMs* with 40 columns. These figures show that our algorithm obtained the best results concerning the runtime and the number of evaluated candidates for artificial *SBDSMs* with a density of 1's under 0.29.

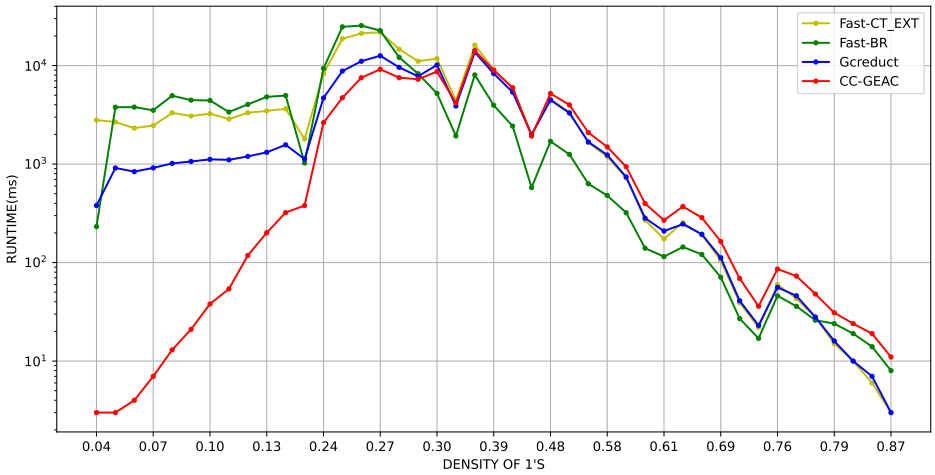

Figure 1. Average runtime in milliseconds (logarithmic scale) vs. *SBDSM*'s density, using artificial matrices with 30 columns.

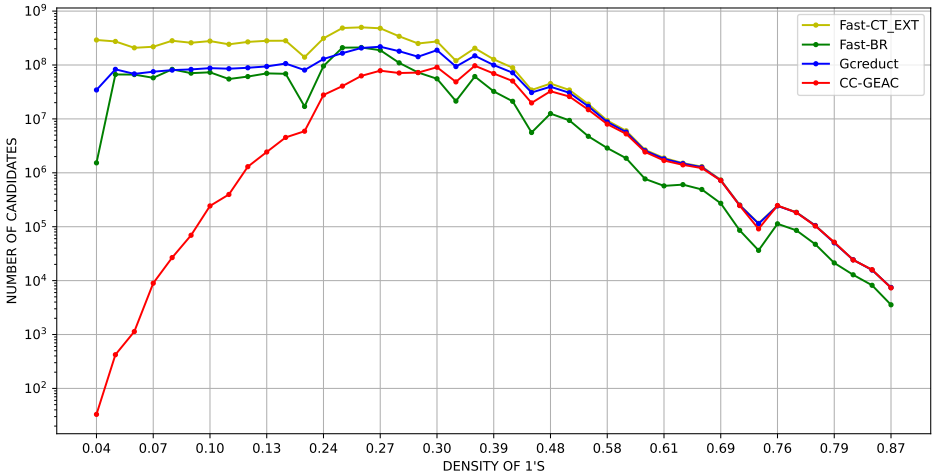

Figure 2. Average number of evaluated candidates (logarithmic scale) vs. *SBDSM*'s density, using artificial matrices with 30 columns.

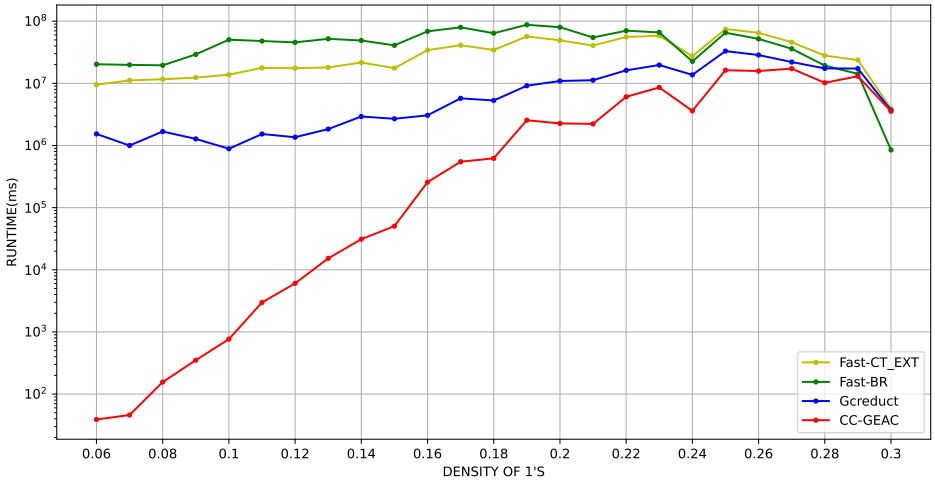

Figure 3. Average runtime in milliseconds (logarithmic scale) vs. *SBDSM*'s density, using artificial matrices with 40 columns.

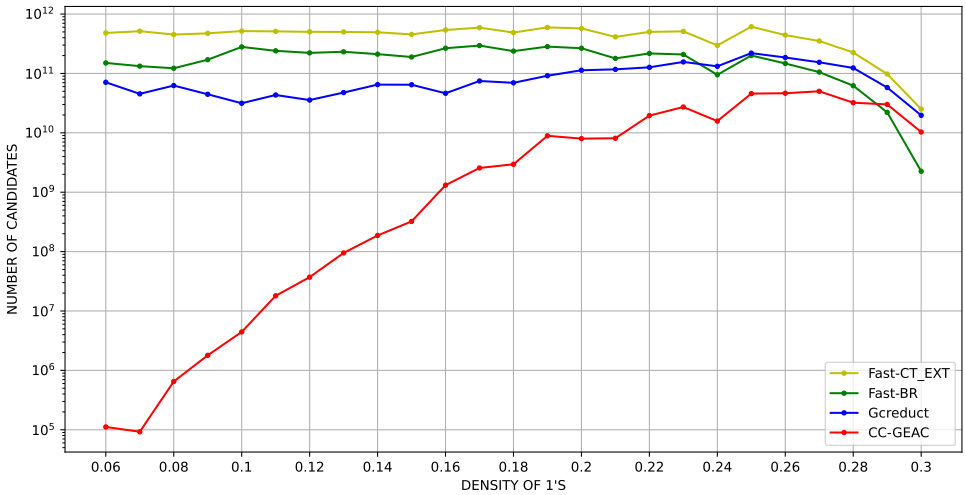

Figure 4. Average number of evaluated candidates (logarithmic scale) vs. *SBDSM*'s density, using artificial matrices with 40 columns.

6. Conclusions

As discussed in this paper, the problem of computing constructs has been little studied, with only a few studies proposing methods to compute constructs using algorithms for reducts computation. This paper introduces a new algorithm, *CC-GEAC*, to compute constructs using the binary simplified discernibility-similarity matrix. Our algorithm utilizes the kernel and the minimum set of attributes to prune the search space, and it searches for gaps in the subsets for further pruning. Additionally, it looks for rows of zeros in the current subset, enabling the elimination of subsets that will never form constructs. We evaluated *CC-GEAC* and compared it, in terms of runtime and the number of evaluated candidates, against the option of computing constructs using algorithms for reducts computation [8]. Our experiments on real and artificial matrices demonstrate that *CC-GEAC* is the best option for computing constructs when the discernibility–similarity matrices have densities lower than 0.29. In most cases, our algorithm evaluates fewer attribute subsets than Fast-CT-EXT [11], FastBR [12], and GCreduct [10], which are the fastest state-of-the-art algorithms for reducts computation that can also be used to compute constructs following [8]. However, this efficiency comes at a higher computational cost. Therefore, in future work, we will work on speeding up the process of evaluating candidates. Another line of future work will involve implementing *CC-GEAC* using GPU-based architectures.

Author Contributions: Y.G.-D., the corresponding author of the contribution "An algorithm for computing all rough set constructs for dimensionality reduction", hereby confirm on behalf of myself and my co-authors (J.F.M.-T., J.A.C.-O. and M.S.L.-C.) that they have participated sufficiently in the work to take public responsibility for its content. This includes participation in the concept, design, analysis, writing, or revision of the manuscript. Accordingly, the specific contributions made by each author are as follows: J.F.M.-T., J.A.C.-O. and M.S.L.-C.: conceptualization, methodology, reviewing, and editing. Y.G.-D.: investigation, software, data curation, writing—original draft preparation, writing—reviewing and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The datasets used during the current study are available in the "UC Irvine Machine Learning Repository": https://archive.ics.uci.edu/ml/datasets.php, "OpenML": https://www.openml.org/; the source code of the CC-GEAC algorithm is available at https://github.com/ygdiaz1202/CC-GEAC.git.

Acknowledgments: The first author thanks the support given by CONAHCYT through his doctoral fellowship.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

RST	rough set theory
SAT	Boolean satisfiability problem
DS	decision system
BDSM	binary discernibility–similarity matrix
SBDSM	simplified binary discernibility–similarity matrix
MSA	minimum attribute subset
CM	cumulative mask
CONAHCYT	Consejo Nacional de Humanidades, Ciencias y Tecnologías

References

1. Pawlak, Z. Rough sets. *Int. J. Comput. Inf. Sci.* **1982**, *11*, 341–356. [CrossRef]
2. Susmaga, R. Reducts and constructs in classic and dominance-based rough sets approach. *Inf. Sci.* **2014**, *271*, 45–64. [CrossRef]
3. Kopczynski, M.; Grzes, T. FPGA supported rough set reduct calculation for big datasets. *J. Intell. Inf. Syst.* **2022**, *59*, 779–799. [CrossRef]
4. Bazan, J.G.; Skowron, A.; Synak, P. Dynamic reducts as a tool for extracting laws from decisions tables. In Proceedings of the Methodologies for Intelligent Systems, Charlotte, NC, USA, 16–19 October 1994; Raś, Z.W., Zemankova, M., Eds.; Springer: Berlin/Heidelberg, Germany, 1994; pp. 346–355. [CrossRef]
5. Stańczyk, U. Application of rough set-based characterisation of attributes in feature selection and reduction. In *Advances in Selected Artificial Intelligence Areas: World Outstanding Women in Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 35–55. [CrossRef]
6. Carreira-Perpinán, M.A. *A Review of Dimension Reduction Techniques*; Technical Report CS-96-09; Department of Computer Science, University of Sheffield: Sheffield, UK, 1997; Volume 9, pp. 1–69.
7. Susmaga, R.; Słowiński, R. Generation of rough sets reducts and constructs based on inter-class and intra-class information. *Fuzzy Sets Syst.* **2015**, *274*, 124–142. [CrossRef]
8. Lazo-Cortés, M.S.; Carrasco-Ochoa, J.A.; Martínez-Trinidad, J.F.; Sanchez-Diaz, G. Computing constructs by using typical testor algorithms. In Proceedings of the Mexican Conference on Pattern Recognition, Mexico City, Mexico, 24–27 June 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 44–53. [CrossRef]
9. Bao, Y.; Du, X.; Deng, M.; Ishii, N. An efficient method for computing all reducts. *Trans. Jpn. Soc. Artif. Intell.* **2004**, *19*, 166–173. [CrossRef]
10. Rodríguez-Diez, V.; Martínez-Trinidad, J.F.; Carrasco-Ochoa, J.A.; Lazo-Cortés, M.S. A new algorithm for reduct computation based on gap elimination and attribute contribution. *Inf. Sci.* **2018**, *435*, 111–123. [CrossRef]
11. Sánchez-Díaz, G.; Piza-Dávila, I.; Lazo-Cortés, M.; Mora-González, M.; Salinas-Luna, J. A fast implementation of the CT_EXT algorithm for the testor property identification. In Proceedings of the Mexican International Conference on Artificial Intelligence, Pachuca, Mexico, 8–13 November 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 92–103. [CrossRef]
12. Lias-Rodriguez, A.; Sanchez-Diaz, G. An algorithm for computing typical testors based on elimination of gaps and reduction of columns. *Int. J. Pattern Recognit. Artif. Intell.* **2013**, *27*, 1350022. [CrossRef]
13. Susmaga, R. Reducts versus constructs: An experimental evaluation. *Electron. Notes Theor. Comput. Sci.* **2003**, *82*, 239–250. [CrossRef]
14. Susmaga, R. Experiments in incremental computation of reducts. *Methodol. Appl.* **1998**. Available online: https://cir.nii.ac.jp/crid/1572824500035732992 (accessed on 1 September 2021).
15. Greco, S.; Matarazzo, B.; Slowinski, R. Rough sets theory for multicriteria decision analysis. *Eur. J. Oper. Res.* **2001**, *129*, 1–47. [CrossRef]
16. Martínez-Trinidad, J.F.; Guzman-Arenas, A. The logical combinatorial approach to pattern recognition, an overview through selected works. *Pattern Recognit.* **2001**, *34*, 741–751. [CrossRef]
17. Lazo-Cortés, M.S.; Martínez-Trinidad, J.F.; Carrasco-Ochoa, J.A.; Sanchez-Diaz, G. On the relation between rough set reducts and typical testors. *Inf. Sci.* **2015**, *294*, 152–163. [CrossRef]
18. Felix, R.; Ushio, T. Rough sets-based machine learning using a binary discernibility matrix. In Proceedings of the Second International Conference on Intelligent Processing and Manufacturing of Materials. IPMM'99 (Cat. No.99EX296), Honolulu, HI, USA, 10–15 July 1999; Volume 1, pp. 299–305. [CrossRef]
19. Skowron, A.; Rauszer, C. The Discernibility Matrices and Functions in Information Systems. In *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory*; Słowiński, R., Ed.; Springer: Dordrecht, The Netherlands, 1992; pp. 331–362. [CrossRef]
20. Santiesteban Alganza, Y.; Pons Porrata, A. LEX: Un nuevo algoritmo para el cálculo de los testores típicos. *Cienc. Mat.* **2003**, *21*, 85–95.

21. Lias-Rodríguez, A.; Pons-Porrata, A. BR: A new method for computing all typical testors. In Proceedings of the Iberoamerican Congress on Pattern Recognition, Guadalajara, Jalisco, Mexico, 15–18 November 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 433–440. [CrossRef]
22. Piza-Dávila, I.; Sánchez-Díaz, G.; Lazo-Cortés, M.S.; Villalón-Turrubiates, I. An Algorithm for Computing Minimum-Length Irreducible Testors. *IEEE Access* **2020**, *8*, 56312–56320. [CrossRef]
23. Dua, D.; Graff, C. *UCI Machine Learning Repository*; University of California, School of Information and Computer Science: Irvine, CA, USA, 2019. Available online: http://archive.ics.uci.edu/ml (accessed on 1 September 2021).
24. Derrac, J.; Garcia, S.; Sanchez, L.; Herrera, F. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. Mult. Valued Log. Soft Comput.* **2015**, *17*, 255–287.
25. Van Rijn, J.N.; Bischl, B.; Torgo, L.; Gao, B.; Umaashankar, V.; Fischer, S.; Winter, P.; Wiswedel, B.; Berthold, M.R.; Vanschoren, J. OpenML: A collaborative science platform. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Prague, Czech Republic, 23–27 September 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 645–649. [CrossRef]
26. Rajalakshmi, A.; Vinodhini, R.; Bibi, K.F. Data Discretization Technique Using WEKA Tool. *Int. J. Sci. Eng. Comput. Technol.* **2016**, *6*, 293–298.