BUAP

Development of fast algorithms for reduct computation

by

Vladímir Rodríguez Diez

Dissertation submitted in partial fulfillment of the requirements for the degree of

PhD. in Computer Science

Advisors:

PhD. J. Arturo Olvera López
Language & Knowledge Engineering
BUAP, Mexico

PhD. J. Francisco Martínez Trinidad
Coordination of Computer Science
INAOE, Mexico

PhD. Manuel S. Lazo Cortés
TecNM/Instituto Tecnológico de Tlalnepantla
Mexico

Language & Knowledge Engineering
Faculty of Computer Science
Av. San Claudio y 14 Sur
Ciudad Universitaria
C.P. 72570

Puebla, Pue. Diciembre, 2022

LKE
Language & Knowledge Engineering

Abstract

Rough Set Theory deals with imperfect knowledge in machine learning. Within Rough Set Theory, reducts are minimal subsets of attributes that preserve the discernibility power of the complete set of attributes in a dataset. Reducts are specially useful as an attribute reduction technique for classification and data storage. Unfortunately, computing all reducts of a dataset has exponential complexity regarding the number of attributes. Therefore, we proposed here a hardware approach for computing all reducts. The proposed platform outperforms previous hardware and software implementations in terms of runtime. In addition, a new algorithm for computing all reducts that uses simple operations for candidate evaluation, which is the fastest algorithm for an specific kind of datasets, was introduced. Furthermore, an experimental study for finding a relation between some properties of a dataset and the fastest algorithms for computing reducts, is presented. This study provides a guide for determining the appropriated algorithm for a specific problem. We proposed finally a new algorithm for computing all the shortest reducts. This new algorithm outperforms all other state–of–the–art algorithms for most datasets. In this case, a guide for selecting a priori the fastest algorithm for computing all the shortest reducts of a specific dataset, was also obtained.

Resumen

La teoría de los conjuntos rugosos es una teoría relativamente nueva para tratar con conocimiento imperfecto. En esta teoría, los reductos son subconjuntos de atributos minimales con la misma capacidad de discernir entre clases del total de atributos en el conjunto de datos. Los reductos son útiles como una técnica de reducción de atributos para la clasificación y el almacenamiento de datos. Desafortunadamente, calcular todos los reductos de un conjunto de datos es un problema de complejidad exponencial. Por este motivo, en esta investigación doctoral, se propone una plataforma de hardware que permite calcular todos los reductos. La plataforma propuesta es más rápida que las plataformas anteriormente reportadas para calcular todos los reductos. Además, se introdujo un nuevo algoritmo para calcular todos los reductos que usa operaciones simples para la evaluación de candidatos, con lo cual logra ser el más rápido para un tipo específico de conjunto de datos. También se presenta un estudio experimental para encontrar una relación entre algunas características del conjunto de datos y los algoritmos más rápidos para calcular todos los reductos. Este estudio provee una guía para seleccionar a priori el algoritmo más rápido para un conjunto de datos específico. Finalmente, proponemos un nuevo algoritmo para calcular todos los reductos más cortos. Este nuevo algoritmo es más rápido que todos los algoritmos reportados anteriormente para este mismo problema, en la mayoría de los conjuntos de datos. En este caso, se obtuvo también una guía para seleccionar a priori el algoritmo más rápido dado un conjunto de datos específico.

Contents

List of Figures

List of Tables

Acronyms

BM	Basic matrix
FPGA	Field Programmable Gate Array
RST	Rough Set Theory
SAT	Satisfiability problem
TT	Typical testor
UCI	University of California, Irvine

Chapter 1

Introduction

Z. Pawlak presented in 1981 [27] a new mathematical theory to deal with imperfect knowledge in machine learning: Rough Set Theory (RST). Decision systems are modeled in RST as tables of items (rows) described by a set of characteristics (columns). Today's data collection considers any potentially useful aspect (attribute) for future data analysis. Thus, having large decision systems with a huge number of attributes is common. This fact impacts negatively on the performance of machine learning algorithms [26]. Rough set reduct [**?**], defined as a minimal subset of attributes with the discernibility power of the complete set of attributes, is a key concept within RST. Computing all reducts from a decision system is well known as an NP–hard [40] problem. Thus, the application of RST is limited by the number of attributes in the dataset.

To deal with larger decision systems, several approximate algorithms have been proposed [7, 50, 14, 4, 5]. Approximate algorithms generate a subset of reducts or find approximate solutions instead of reducts. An approximate solution is an attribute subset preserving the discernibility capacity of the complete set of attributes, but it includes redundant attributes; or it is an attribute subset that preserves the discernibility capacity to a certain degree. Since approximate algorithms do not return all the reducts of a decision system, the development of exact algorithms for reduct computation is still an active research topic [44, 41, 51, 16, 32].

Computing all reducts requires a high computational effort, and it results, most of the times, in a large amount of reducts. However, in practice, sometimes computing a subset of reducts, that satisfy some additional restrictions [45, 17], is enough. A special case is the computation of all the shortest reducts. This subset is a representative

sample of all reducts [44]. The shortest reducts are specially useful, for instance, in data reduction applications and classification [59]. Unfortunately, the computation of all the shortest reducts is also an NP–hard [40] problem. Thus, the search of new algorithms for computing the set of all the shortest reducts is a challenging research topic.

Throughout this PhD research, a new algorithm for computing all reducts of a decision system as well as a new algorithm for computing all the shortest reducts, are introduced. Our proposed algorithms operate over the basic matrix (a reduced representation of the discernibility information in the decision system). In addition, we have conducted comparative studies with state–of–the–art algorithms. Within this study, the relation between algorithms' performance and some characteristics of the basic matrix is explored. Based on this relation, some guidelines for selecting the fastest algorithm for a given decision system are concluded.

1.1 The Problem

Rough set reducts can discard attributes in a dataset without degrading the discernibility between objects in different classes. Therefore, reduct computation has been an active research topic, specially the shortest reducts [16]. Feature selection by means of rough set reducts is exposed in [57]. In addition, [17] stated that attribute reduction is one of the most relevant applications of rough sets and enumerate several algorithms for computing reducts. We find consensus in the literature about the relevance and the topicality of rough set reducts.

The use of heuristic strategies as those proposed in [58, 7, 15**?**] constitute efficient solutions for computing reducts but they may not find one of the shortest reducts. Algorithms based on stochastic strategies, such as those proposed in [52, 14, 53, 4]

cannot return one of the shortest reducts for sure, as discussed in Chapter 3 of this thesis. Algorithms designed for computing all reducts [41, 44, 51] come as a highly expensive solution for obtaining the shortest reducts.

This research aims the development of new algorithms for obtaining all reducts or all the shortest reducts from a decision systems. These are NP–hard problems, which requires clever pruning strategies for reducing the runtime. The new algorithms proposed in this work must be competitive with the state–of–the–art algorithms in most cases and faster for some kinds of dataset. The experimental data consist of decision systems taken from the UCI machine learning repository [3] as well as synthetic datasets.

1.2 Objectives

The **general objective** of this thesis is obtaining new algorithms for reduct computation in decision systems; that must be the fastest in some kinds of decision systems.

Some characteristics of the basic matrix are evaluated for selecting the appropriate pruning properties for the given problem. Two approaches are explored: the computing all reducts and computing all the shortest reducts.

The **specific objectives** are:

1. Finding a relation between some properties of the basic matrix and the fastest algorithms for computing all reducts.

2. Obtaining a new algorithm for computing all reducts.

3. Finding a relation between some properties of the basic matrix and the fastest algorithms for computing all the shortest reducts.

4. Obtaining a new algorithm for computing all the shortest reducts.

1.3 Contributions

From the literature, we found that several hardware implementations were reported as the fastest solution for computing reducts [46, 47, 48, 13, 19, 49]. Consequently, this PhD research, presents a new hardware–software platform for reduct computation. This platform was developed as a hardware implementation of CT–EXT [36]. Our proposed platform was experimentally compared against a software implementation of CT–EXT, as well as a hardware implementation of the algorithm BT [34]. These experiments serve as an assessment of the feasibility of hardware architectures for reduct computation.

Unfortunately, FPGA resources limit the size of the problem that can be solve with these platforms. Thus, our subsequent research was redirected on to the development of algorithms for reduct computation. Consequently, a new algorithm for computing all reducts of a decision system was proposed. This new algorithm is the fastest alternative for reduct computation on those decision systems whose associated basic matrix has low density of 1's. An experimental comparison using synthetic data for studying the relation between the algorithms' runtime and the density of 1's of the basic matrix was also presented. This relation was experimentally verified over information systems from the UCI machine learning repository.

Furthermore, a new algorithm for computing all the shortest reducts of a decision system was developed. Likewise, an experimental comparison using synthetic data for analyzing the relation between the algorithms' runtime and the density of 1's of the basic matrix was also presented. From this experiment, the proposed algorithm was found to be the fastest alternative for computing the shortest reducts on those decision systems whose associated basic matrix has medium or high density of 1's. This relation was also experimentally verified over information systems from the UCI machine learning repository.

1.4 Thesis Organization

The content of this document are organized as follows. In Chapter 2, some basic concepts from Rough Set Theory are introduced. Chapter 3 presents a review of the state–of–the–art algorithms for reduct computation. In Chapter 4, a new hardware platform for computing all reducts of a decision system is proposed. In Chapter 5, a new algorithm for computing all reducts of a decision system is proposed. In Chapter 6, a new algorithm for computing all the shortest reducts of a decision system is introduced. Chapter 7 presents our conclusions and contributions, as well as the publications related to this research and the future work.

Chapter 2

Basic Concepts

Rough Set Theory (RST) models objects as entities completely described by a set of attributes. RST assumes that two objects are indiscernible if their value of every attribute is the same. The mathematical foundations of RST relay on the indiscernibility relations of objects. In this chapter, the basic concepts of RST are provided.

2.1 Decision System

Information Systems (IS) hold the objects data in RST. A table with rows representing objects while columns represent attributes or features is the usual representation of an IS, it is formally defined as a pair $IS = (U, A)$ where U is a finite non-empty set of objects $U = \{x_1, x_2, ..., x_n\}$ and A is a finite non-empty set of attributes (features, variables). For each element in A there is a mapping: $a : U \rightarrow V_a$. V_a is the *value set* of a. Attributes in A are further divided into condition attributes C and decision attributes D such that $A = C \cup D$, $C \neq \emptyset$ and $C \cap D = \emptyset$. An IS with $D \neq \emptyset$ is called a *Decision System* (DS). Table 2.1 shows an example of a DS.

Table 2.1: Example of a decision system, where $c_0 - c_6$ are condition attributes and d is the decision attribute.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	d
x_1	0	blue	medium	3	12	$= 1$	0	bad
x_2	0	blue	long	3	13	< 1	1	bad
x_3	0	blue	medium	3	20	< 1	1	good
x_4	0	green	medium	2	20	< 1	1	bad
x_5	0	blue	medium	1	20	> 1	1	bad
x_6	0	blue	short	3	20	< 1	1	good
x_7	0	red	medium	3	20	< 1	0	bad
x_8	1	blue	medium	2	20	< 1	1	bad

In the example of Table 2.1, $D = \{d\}$, where d is the decision attribute that determines the class an object belongs to. This example is a decision system with two classes.

Decision attributes partition of the universe U into *decision classes*, which lead to the concept of the *positive region of the decision*. The set $POS_B(d)$, called the *B-positive region of d*, is defined as the set of all objects in U such that if two of them have the same value for every attribute in B, they belong to the same class induced by d.

For the decision system shown in table 2.1, we have:

$$\begin{aligned} POS_{\{c_3\}}(d) &= \{x_4, x_5, x_8\} \\ POS_{\{c_4\}}(d) &= \{x_1, x_2\} \\ POS_{\{c_3,c_4\}}(d) &= \{x_1, x_2, x_4, x_5, x_8\} \end{aligned}$$

2.2 Reducts

The *Indiscernibility Relation* for a subset of condition attributes $B \subseteq C$ is defined as:

$$IND(B|D) = \{(x, y) \in U \times U |\ [d(x) = d(y)] \vee [\forall c \in B\ (c(x) = c(y))]\}$$

where $c(x)$ is the value of the attribute c for the object x, and $d(x)$ is the value of the decision attribute d for the object x. The indiscernibility relation for B is the set of all pairs of objects from different decision classes that cannot be distinguished by considering only the attributes in B, jointly with the set of all pairs of objects that belong to the same decision class.

For decision systems, we want, most of the times, to discern between objects that belong to different classes. For this purpose, it is relevant the concept of *decision reduct*;

which we define, in terms of the indiscernibility relation, as follows:

Definition 2.1 *Let C be the set of condition attributes and $D = \{d\}$ be the set containing the decision attribute d of a decision system DS, the set $B \subseteq C$ is a decision reduct of DS if:*

1. $IND(B|D) = IND(C|D)$.

2. $\forall c \in B, IND(B - \{c\}|D) \neq IND(C|D)$.

Decision reducts have the same capability as the complete set of condition attributes for discerning objects that belong to different classes (Condition 1), and they are minimal with respect to inclusion (Condition 2). We call super–reduct to any set B satisfying Condition 1, regardless of Condition 2. Hereinafter, for simplicity, we will call reducts to the decision reducts.

The intersection of all reducts of a decision system is called the *core*, i.e., the core of a decision system is the set of those attributes that appear into all the reducts.

2.3 The Discernibility Matrix

Discernibility relations are usually stored in a symmetric $|U| \times |U|$ matrix defined as the *discernibility matrix*. Every element m_{ij} from the discernibility matrix DM can be defined as

$$m_{ij} = \begin{cases} \{c \in C : c(x_i) \neq c(x_j)\} & \text{for } \ d(x_i) \neq d(x_j) \\ \emptyset & \text{otherwise} \end{cases} \tag{2.1}$$

Table 2.2 shows the discernibility matrix for the decision system shown in table 2.1 as an upper triangular matrix (empty sets are omitted for clarity).

Table 2.2: Discernibility matrix of the decision system shown in Table 2.1.

$x \in U$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1			$\{c_4, c_5, c_6\}$			$\{c_2, c_4, c_5, c_6\}$		
x_2			$\{c_2, c_4\}$			$\{c_2, c_4\}$		
x_3				$\{c_1, c_3\}$	$\{c_3, c_5\}$		$\{c_1, c_6\}$	$\{c_0, c_3\}$
x_4						$\{c_1, c_2, c_3\}$		
x_5						$\{c_2, c_3, c_5\}$		
x_6							$\{c_1, c_2, c_6\}$	$\{c_0, c_2, c_3\}$
x_7								
x_8								

From DM, the *discernibility function* (f_{DM}) can be defined. This is a Boolean function representing the presence of the corresponding attribute (True) or its absence (False) in DM.

$$f_{DM} = \wedge\{\vee c^*_{ij} : 1 \leq j \leq i \leq |U|, m_{ij} \neq \emptyset\} \tag{2.2}$$

where $\vee c^*_{ij}$ denotes the logical disjunction of all the variables corresponding to the attributes in m_{ij}.

From the discernibility matrix of Table 2.2, we have:

$f_{DM} = (c^*_4 \vee c^*_5 \vee c^*_6) \wedge (c^*_2 \vee c^*_4 \vee c^*_5 \vee c^*_6) \wedge (c^*_2 \vee c^*_4) \wedge (c^*_2 \vee c^*_4) \wedge (c^*_1 \vee c^*_3) \wedge (c^*_3 \vee c^*_5) \wedge (c^*_1 \vee c^*_6) \wedge (c^*_0 \vee c^*_3) \wedge (c^*_1 \vee c^*_2 \vee c^*_3) \wedge (c^*_2 \vee c^*_3 \vee c^*_5) \wedge (c^*_1 \vee c^*_2 \vee c^*_6) \wedge (c^*_0 \vee c^*_2 \vee c^*_3)$

2.3.1 The Basic Matrix

The *Binary Discernibility Matrix* is a binary representation of DM, where columns represent single condition attributes and rows represent pairs of objects belonging to different decision classes. The discernibility element $m(i, j, c)$ for two objects x_i and x_j and a single condition attribute $c \in C$ is given in a binary representation, such that:

$$m(i,j,c) = \begin{cases} 1 & \text{if } c(x_i) \neq c(x_j) \\ 0 & \text{otherwise} \end{cases}$$

Table 2.3 shows the binary discernibility matrix of the decision system shown in Table 2.1.

Table 2.3: Binary Discernibility Matrix of the decision system in Table 2.1.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6
x_1, x_3	0	0	0	0	1	1	1
x_1, x_6	0	0	1	0	1	1	1
x_2, x_3	0	0	1	0	1	0	0
x_2, x_6	0	0	1	0	1	0	0
x_4, x_3	0	1	0	1	0	0	0
x_4, x_6	0	1	1	1	0	0	0
x_5, x_3	0	0	0	1	0	1	0
x_5, x_6	0	0	1	1	0	1	0
x_7, x_3	0	1	0	0	0	0	1
x_7, x_6	0	1	1	0	0	0	1
x_8, x_3	1	0	0	1	0	0	0
x_8, x_6	1	0	1	1	0	0	0

The *Simplified Binary Discernibility Matrix* is a reduced version of the binary discernibility matrix after applying absorption laws. In Testor Theory [20], this concept is called *Basic Matrix*. Hereinafter, we will refer to this simplified representation as basic matrix.

Table 2.4: Basic Matrix of the Binary Discernibility Matrix shown in Table 2.3.

c_0	c_1	c_2	c_3	c_4	c_5	c_6
0	0	0	0	1	1	1
0	0	1	0	1	0	0
0	1	0	1	0	0	0
0	0	0	1	0	1	0
0	1	0	0	0	0	1
1	0	0	1	0	0	0

Definition 2.2 *Let BDM be a binary discernibility matrix and r_k be a row of BDM. r_k is a superfluous row of BDM if there exists a row r in BDM such that $\exists i|(r[i] < r_k[i]) \wedge \forall i|(r[i] \leq r_k[i])$, where $r[i]$ is the i-th element of the row r.*

The basic matrix is obtained by removing every superfluous row from the binary discernibility matrix. Table 2.4 shows the basic matrix from the binary discernibility matrix of Table 2.3. An important fact is that all reducts of a decision system, can be computed from this reduced matrix [56].

Chapter 3

Related work

This chapter presents a review of the related work reported in the literature for reduct computation. Given the close relation between the concept of reduct from RST and the concept of typical testor from Testor Theory [6], algorithms developed for typical testor computation can be applied to reduct computation [22]. Thus, the most relevant algorithms reported for typical testor computation are also included in this chapter.

The content of this chapter are structured as follows: Section 3.1 presents the most relevant approximate algorithms for computing reducts. In Section 3.2, exact algorithms of the state–of–the–art for computing reducts are discussed. In Section 3.3, some of the most relevant hardware architectures reported for computing reducts are presented. Finally, in Section 3.4, our concluding remarks on this literature review are given.

3.1 Approximate Algorithms for Reduct Computation

We present a taxonomy of approximate algorithms for computing reducts in Figure 3.1.

3.1.1 Algorithms for Computing a Single Reduct

QUICKREDUCT [7] is an algorithm that adds, one at a time, the attribute with the highest significance. The significance of an attribute is evaluated as the number of objects included into the positive region when the attribute is considered. This is one of the earliest approaches to approximate computation of a single reduct.

The algorithms reported in [50] (RA–Entropy, RA–Roughness and RA–QS–Elements)

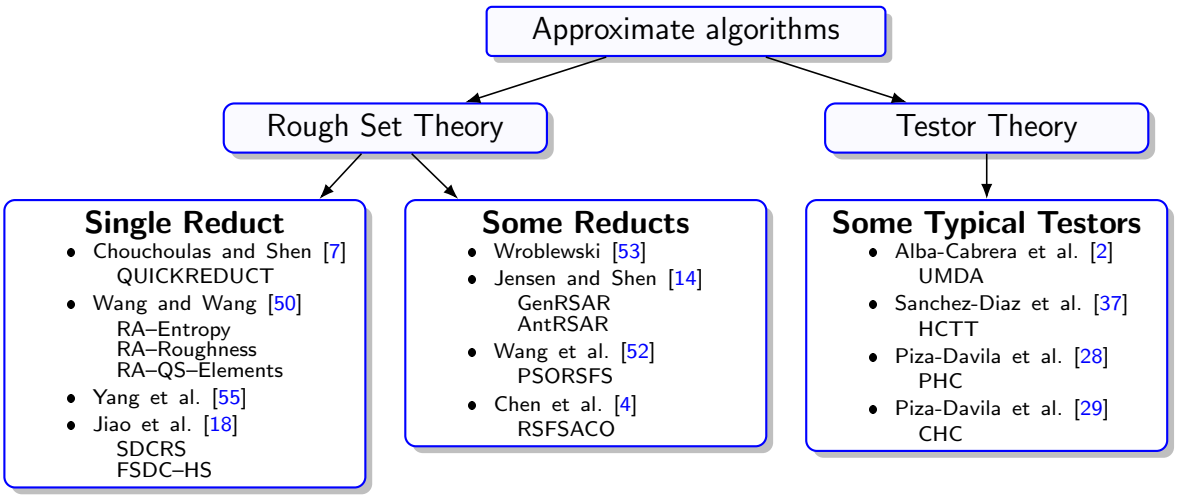

Figure 3.1: Taxonomy of approximate algorithms for computing reducts.

use different strategies for evaluating the attribute significance, and operate over the discernibility matrix. However, in their search strategy, they are similar to QUICKREDUCT.

The use of binary cumulative operations for evaluating candidate subsets over the binary discernibility matrix was introduced in [55]. This is also a greedy algorithm, similar to QUICKREDUCT in its search strategy and the evaluation of attribute significance. However, the new candidate evaluation process reduces the runtime required for computing a single reduct, according to their experiments.

The method proposed in [18] subdivides the dataset to reduce the runtime of computing a single reduct. The complete decision system is divide into a master–table and several smaller sub–tables. Finally, the results are merged to obtain the reducts of the original decision system. Two algorithms are proposed (SDCRS and FSDC–HS) using different subdivision strategies.

3.1.2 Algorithms for Computing Some Reducts

Several stochastic algorithms have been developed to compute locally shortest reducts. These algorithms may find many reducts in a short time but global shortest reducts may

not be included in the result. The genetic algorithms proposed in [53] encode candidates as bit strings with a positional representation of attributes. The fitness functions depend on the number of attributes in the candidate, penalizing those candidates with a large number of attributes. A second optimization parameter is the number of pairs of objects that can be discerned by the given candidate. A good point of genetic algorithms is that the use of a fitness function leads the search down to a set of reducts with the desired properties. GenRSAR [14] is a simple algorithm that also uses a genetic search strategy in order to find reducts, but operates over the discernibility function.

Other bio-inspired approaches to reduct computation are based on Ant Colony Optimization (ACO): AntRSAR [14], RSFSACO [4]; and Particle Swarm Optimization (PSO): PSORSFS [52]. These stochastic approaches do not require complex operators such as mutation and crossover used in genetic algorithms. On the contrary, ACO and PSO relay on simple mathematical operations, which lead to shorter runtimes.

3.1.3 Algorithms for Computing Some Typical Testors

From Testor Theory, in [2] the UMDA algorithm was proposed, which is based on the Univariate Marginal Distribution Algorithm as the core of the search strategy. This approach is a fast alternative to genetic algorithms for computing a subset of typical testors (reducts).

In [37] a Hill-Climbing algorithm (HCTT) that incorporates an acceleration operation at the mutation step was introduced. This new algorithm provides a more efficient exploration of the search space than previous stochastic algorithms for computing a subset of typical testors. In [28] a parallel acceleration (PHC) of HCTT, taking advantage of the intrinsic parallelism of this kind of algorithms was proposed. Later, in [29] a CUDA–based implementation (CHC) of these algorithms was presented. CHC takes advantage of GPU computing for speeding up the parallel search process of PHC.

3.2 Exact Algorithms for Reduct Computation

In Figure 3.2, we propose a taxonomy of the reported exact algorithms for computing reducts. We use the taxonomy shown in Figure 3.2 to organize the contents of this section.

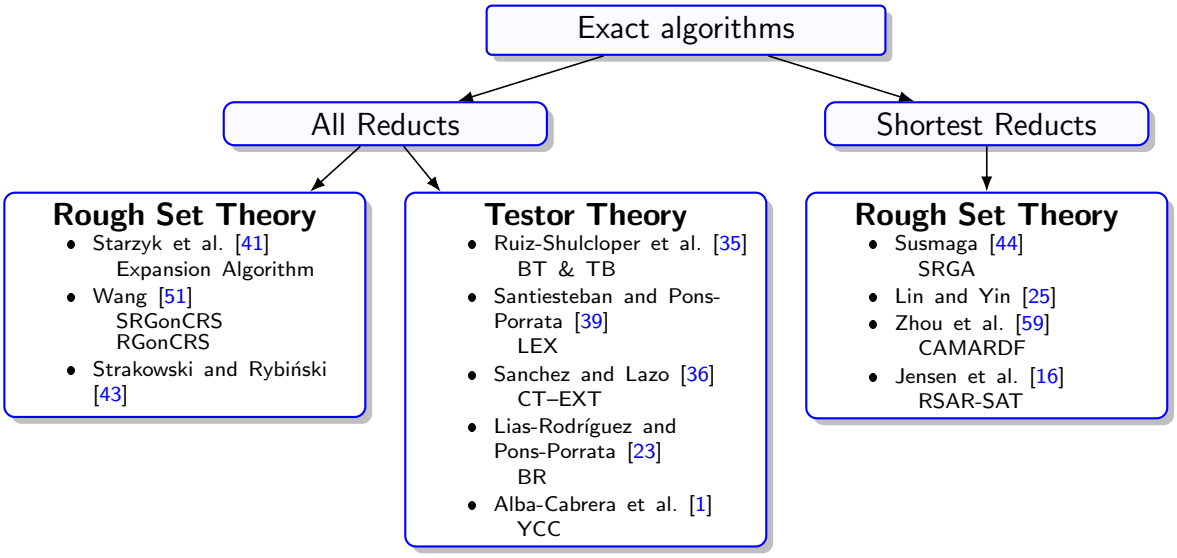

Figure 3.2: Taxonomy of exact algorithms for computing reducts.

3.2.1 Algorithms for Computing all Reducts

An early method for computing all reducts of a decision system (Expansion Algorithm) was proposed in [41, 42]. This is a divide and conquer approach. On each step, the absorption laws are applied over the incoming discernibility matrix to obtain a basic matrix. Then, the redundant attributes are compressed at each recursion level. One of the most discerning attributes is selected (in the same way as QUICKREDUCT) and the problem is divided into two sub–problems:

- Finding reducts containing the selected attribute. Thus a recursive function is called with a new basic matrix, having only those rows where the selected attribute

does not appear.

- Finding reducts that do not contain the selected attribute. Thus a recursive function is called with a new basic matrix, removing the column corresponding to the selected attribute.

The base case in the recursion is reached when each attribute in the incoming discernibility matrix appears in a single row. In this way, a set of super–reducts is obtained and all supersets must be removed in order to obtain the reduct set.

RGonCRS [51] is an algorithm for computing all reducts. This is a very elaborated approach that works over the dataset instead of the basic matrix. First, RGonCRS computes the core, and then searches for reducts as supersets of the core. It has a recursive implementation, where contributing attributes are selected as in QUICKREDUCT. In [51] a second algorithm called SRGonCRS is proposed, which subdivides the dataset and computes the reducts incrementally.

Different variants (DT, DISC FUNCTION and CANDIDATE REDUCTS) for subdividing a reduct computation problem were proposed and discussed in [43]. They also discuss the conditions for subdividing the problem, and introduce a criterion for selecting the best subdivision method.

3.2.2 Algorithms for Computing all Typical Testors

One of the first works on the computation of all typical testors (TT) was presented in [35]. In this work, two similar algorithms were presented (BT and TB). These algorithms codify a subset of features as a binary word with as many bits as features in the dataset. A 0 represents the absence of the corresponding feature in the current subset while a 1 represents its inclusion. In this way, candidate subsets are evaluated in the natural ascending order induced by the binary numbers (BT) or the reverse order

(TB). The pruning of the search space is based on the minimality condition of a TT and a convenient arrangement of the basic matrix associated to the dataset. Finally, testors found by these algorithms must be filtered in order to remove any non typical testor.

The main ideas behind LEX [39] are a new traversing order of candidates (which resembles the lexicographical order) and the concept of gap. In LEX, the typical condition is verified first and only for those potentially TT, the testor condition is checked. The concept of gap allows to avoid the evaluation of subsets of a candidate that is a TT or a non–testor and it includes the last feature of the dataset, in the traversing order.

CT–EXT [36], is an algorithm for computing all TT. Following a traversing order similar to that in LEX, this algorithm searches for testors without verifying the typical condition. This way, in comparison to LEX, a larger number of candidates are evaluated, but the cost of each evaluation is lower. Results from experiments show that CT–EXT is faster than all previous algorithms in most datasets. Afterwards, BR [23], a Recursive algorithm based on Binary operations. BR is similar to LEX, but its recursive nature encloses a great improvement. Given a candidate subset, the remaining features are tested a priori and those being rejected are excluded from subsequent evaluations. In [38] a cumulative procedure for the CT–EXT algorithm was presented. This fast-CT_EXT implementation drastically reduces the runtime for most datasets at no extra cost. In [24] the gap elimination and column reduction are added to BR. This fast-BR algorithm is, no doubt, the one evaluating the minimum amount of candidates in the state–of–the–art. The main drawback of fast–BR and BR is, as in LEX, the high cost of evaluating the typical condition for each candidate.

YYC [1], is another testor–finding algorithm. This algorithm computes the typical testors incrementally over the rows of the basic matrix. Although they claim that this algorithm verifies less candidates than previous alternatives, two weak points should

be addressed. First, BR was not included in their comparisons; and second, in YYC the evaluation cost for a candidate is high compared to previous algorithms. YYC verifications involve computing the Hamming weight.

3.2.3 Algorithms for Computing all the Shortest Reducts

In [44], after a strong discussion on reduct computation and the role of the shortest reducts within RST applications, two algorithms were presented. One for computing all k-reducts (reducts with length no greater than k), and another one to compute all the shortest reducts (SRGA). Both algorithms were built on top of the Modified Reduct Generation Algorithm (MRGA), which is the core proposal of the author. MRGA introduces the application of absorption laws over the discernibility function, which is a representation of the discernibility information; this allows reducing the runtime in comparison to previous algorithms. However, in this approach, every candidate is evaluated looking for superfluous attributes. This is an operation with a high computational cost, which reduces the performance of the algorithm in some cases.

In [25], a heuristic approach to reduce the runtime of computing the shortest reducts is presented. This heuristic consists in finding a single short reduct and then, pruning the search space by considering only those attribute combinations with no higher length. The main drawback of this algorithm is that the second step searches for reducts with no additional pruning strategies; thus it explores all possible attribute combinations, which is infeasible in most practical cases.

The algorithm proposed in [59] (CAMARDF) operates over the reduced discernibility function which is a representation of the discernibility information after applying absorption laws. Furthermore, CAMARDF sorts the attributes to be processed at each recursion level by their significance. In this way, those attributes that discern more pairs of objects confused by the current candidate, are added first. This strategy reduces the

search space; however, computing the attribute significance has a high computational cost.

Although originally intended for computing a single minimal reduct, the algorithm proposed in [16] (RSARSAT) can be modified in order to obtain all the shortest reducts of a decision system. The method introduced in this work reduces the problem of finding a reduct from the discernibility function to the SAT problem [9].

3.3 Hardware Architectures for Reduct Computation

In Figure 3.3, we propose a taxonomy of the reported hardware architectures for computing reducts. We use the taxonomy shown in Figure 3.3 to organize the contents of this section.

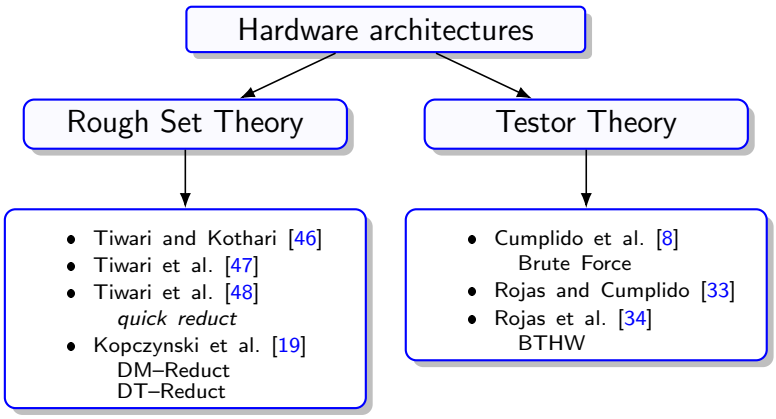

Figure 3.3: Taxonomy of hardware architectures for computing reducts.

3.3.1 Hardware Architectures for Computing Reducts

A parallel acceleration of the algorithm presented in [55], for reduct computation from a binary discernibility matrix, was developed in [46, 47]. This FPGA implementation computes a single reduct for object identification into an intelligent robot. In [48] a

quick reduct algorithm, similar to that presented in [7], is proposed and implemented in a hardware fashion. In [49] a thorough survey of FPGA applications in rough set reduct computation is presented.

The authors of [19] proposed two hardware architectures for single reduct computation: DM–Reduct, which operates over the discernibility matrix; and DT–Reduct, which computes reduct taking a decision table as input. Although the authors claim that a huge acceleration is achieved, the experiments presented in [19] to validate their results are performed just over a small dataset; an this does not imply its applicability to larger cases where such acceleration is actually needed.

3.3.2 Hardware Architectures for Computing Typical Testors

From the Testor Theory, several attempts by means of FPGA implementations, to overcome the complexity of the problem of computing all typical testors have been reported. In a first work [8], an FPGA–based brute force approach for computing testors was proposed. This first approach did not take advantage of dataset characteristics to reduce the number of candidates to be tested; thus all 2^n combinations of n features have to be tested. Then, in [33] a hardware architecture implementing the BT algorithm for computing typical testors was introduced. These two works compute a set of testors on the FPGA device while the typical condition has to be evaluated afterwards by the software component in the hosting PC. Thus, in [34] a hardware–software platform for computing typical testors that implements the BT algorithm, similar to [33], was proposed; but it also includes a new module that eliminates most of the non typical testors before transferring them to a host software application for a final filtering.

3.4 Concluding Remarks

From our literature review, we noticed that the development of algorithms from Testor Theory is biased to find all typical testors of a decision system. Algorithms from Rough Set Theory, on the other hand, are mainly divided into three categories:

- Algorithms for computing a pseudo–optimal reduct according to a criterion (which is, most of the time, the length of the obtained reduct).
- Algorithms for computing one of the shortest reducts.
- Algorithms for computing all reducts.

The most prolific research area in Rough Set Theory is the development of algorithms for computing a pseudo–optimal reduct.

We found that algorithms for computing all reducts [42, 51] and algorithms for computing all the shortest reducts [44, 59] have several disadvantages since they do not work over the basic matrix, and they use complex data representations. Most of the ideas for pruning the search space in these algorithms can be found in Testor Theory as well, although they are expressed with a different nomenclature.

Hardware architectures for computing reducts are mainly focused on computing a single pseudo–optimal reduct [46, 47, 48, 13, 19, 49]. However, this problem is out of the scope of this PhD research.

Algorithms for computing typical testors operate over the basic matrix. It is important to highlight that computing the basic matrix from the original dataset has quadratic complexity regarding the number of objects in the dataset, while computing all typical testors has exponential complexity regarding the number of attributes. Therefore, in most large datasets, it is better to work over the basic matrix. Properties used by these algorithms are implemented by means of Boolean operations and bit ma-

nipulations, which lead to faster implementations. We identified fast–CT–EXT [38] and fast–BR [24] as the fastest algorithms reported in the literature for computing typical testors.

Hardware architectures reported in Testor Theory [8, 33, 34] are focused on computing all typical testors, and they make a great runtime reduction regarding software implementations.

Chapter 4

Hardware Architectures for Speeding up Reduct Computation

From our preliminary literature review, we found that hardware implementations have been reported as the fastest approach to reduct computation [46, 47, 48, 13, 19, 49]. In addition, from Testor Theory, several FPGA–based platforms for computing all typical testors (reducts) have been also reported [8, 33, 34]. For this reason, our first step was developing a hardware implementation of the CT–EXT algorithm [36]; which is one of the fastest algorithms for typical testor (reduct) computation reported in the literature.

The rest of this chapter is structured as follows. In Section 4.1, we present a brief description of the CT–EXT algorithm. Section 4.2 introduces the proposed hardware platform. The evaluation of the proposed platform and a discussion of the experimental results are presented in Section 4.3. Finally, Section 4.4 shows our concluding remarks and the directions for the rest of this PhD research.

4.1 CT–EXT algorithm

We first present some definitions and propositions supporting the CT–EXT algorithm.

Let $[c|Y]$ denote an ordered list of attributes such that c is the first attribute in the list and Y is the ordered list of the remaining attributes. If we have, for instance $[c_1, c_3, c_6]$, we can express it as $[c_1|[c_3, c_6]]$ according to this notation. In the same way, $[c_3, c_4]$ can be expressed as $[c_3|[c_4]]$. The empty list is denoted as [ ].

Let us also define an order relation $\prec$ over the set of ordered lists of attributes as follows.

1. Let $Y = [c_i|Y']$ and $Z = [c_j|Z']$ be two ordered lists of attributes. Then, $Y \prec Z$ if $i < j$.

2. Let $Y = [c_i|Y']$ and $Z = [c_i|Z']$ be two ordered lists of attributes. Then, $Y \prec Z$ if $Y' \prec Z'$.

3. $\forall Y : [\,] \prec Y$.

Hereinafter, we will refer to this order as the lexicographical order. We denote the length of a list L as $|L|$. Furthermore, we will use $+$ to denote list concatenation such that:

$$[c_1, c_3, c_4] + [c_6, c_7] = [c_1, c_3, c_4, c_6, c_7]$$

The following definition of super–reduct is equivalent to Condition 1 from Definition 2.1 [22]. In this section, we will use this definition for making clearer the explanation of CT–EXT.

Definition 4.1 *Let BM be a basic matrix and L be an ordered list of condition attributes. L is associated to a super–reduct iff in the sub–matrix of BM considering only the attributes in L, there is no zero row (a row with only zeros).*

Now we define the concept of *contribution* which is a fundamental component in most algorithms for reduct and typical testor computation [36, 51, 23, 32].

Definition 4.2 *Let BM be a basic matrix, L be an ordered list of attributes and $c_i \in C$ be an attribute, such that $c_i \notin L$. c_i contributes to L iff the number of zero rows, in the sub-matrix of BM considering the attributes in $L + [c_i]$, is lower than considering only the attributes in L.*

From this concept, the following proposition was stated and proved in [36].

Proposition 4.1 *Let L be an ordered list of attributes and $c_i \in C$ be an attribute such that $c_i \notin L$. If c_i does not contribute to L, then $L + [c_i]$ cannot be associated to a subset of any reduct.*

Using Proposition 4.1, the evaluation of supersets of a candidate associated to $L+[c_i]$ can be avoided if c_i does not contribute to L.

In order to reduce the search space, CT–EXT arranges the basic matrix as follows: first, one of the rows of the basic matrix with the fewest number of 1's is selected. Then, the selected row is moved to the top, and all columns in which it has 1, are moved to the left. Table 4.1 shows the basic matrix from Table 2.4, after performing the above explained arrangement.

Table 4.1: Arranged Basic Matrix computed from Table 2.4.

c_3	c_0	c_1	c_2	c_4	c_5	c_6
c'_0	c'_1	c'_2	c'_3	c'_4	c'_5	c'_6
1	1	0	0	0	0	0
1	0	1	0	0	0	0
1	0	0	0	0	1	0
0	0	1	0	0	0	1
0	0	0	1	1	0	0
0	0	0	0	1	1	1

Using the above described arrangement, Proposition 4.2 states that if we follow the lexicographical order for traversing the search space and we reach an ordered list of attributes L satisfying $c_{i_0} = 0$, the search can be stopped [23].

Proposition 4.2 *Let BM be an arranged basic matrix and $L = [c_{i_0}, ..., c_{i_s}]$ be an ordered list of attributes. If the first row in the column corresponding to c_{i_0} has 0, denoted as $c_{i_0}[0] = 0$, L is not associated to a super–reduct. Furthermore, there is no list L', such that $L \prec L'$, associated to a super–reduct.*

Algorithm 4.1 shows the pseudocode of CT–EXT, a detailed explanation of this algorithm can be seen in [36]. The function Evaluate(BM, B) returns three values:

super_reduct, *reduct* and *zero_rows*. *super_reduct* is TRUE if the set B is a super–reduct of BM and FALSE otherwise. *reduct* is TRUE if the set B is a reduct and FALSE otherwise. *zero_rows* is the amount of zero rows of B. The function LastOne(B) returns the position of the rightmost element in the set B.

4.2 Proposed platform

A common stage in all algorithms for computing reducts is the verification of each candidate combination over the basic matrix. This is an intrinsic parallel operation that a hardware implementation could take advantage of. The time complexity of evaluating a candidate for the super–reduct condition is $O(nm)$ and for the minimality condition is $O(n^2m)$; where n is the number of attributes and m is the number of rows in the basic matrix. In the proposed hardware component, these conditions are simultaneously evaluated in a single clock cycle. Moreover, the main novelty of the proposed platform relays on the Candidate Generator module. This new module implements the lexicographical order [36] as in the CT–EXT algorithm. This traversing order allows our proposal to evaluate less candidates than those evaluated by other hardware architectures reported in the literature; reducing, in this way, the runtime.

The proposed platform is shown in Figure 4.1. The platform comprises a host PC and an Atlys board populated with a FPGA Spartan-6 device [11]; which are connected through a USB cable. A custom developed software application, running in the PC, handles all the processes needed to create the bitstream file to configure the FPGA device. The custom architecture implemented in the FPGA carries out all the calculations needed to generate the reducts and sends the results to the PC where the user can then analize the results. A detailed description of all platform components are given below.

Algorithm 4.1 CT-EXT algorithm

```
1: Input: BM - sorted basic matrix with m rows and n columns.
2: Output: RS - set of reducts.
3: RS ⇐ {}
4: j ⇐ 0                                          ▷ first attribute from BM to be analyzed
5: while BM[0, j] ≠ 0 do
6:     B ⇐ {c_j}                                  ▷ current attribute subset
7:     super_reduct, reduct, zero_rows ⇐ Evaluate(BM, B)
8:     if super_reduct = TRUE then
9:         if reduct = TRUE then                  ▷ B is a reduct
10:            RS ⇐ RS ∪ B
11:    else
12:        i ⇐ j + 1
13:        while i < n do
14:            B ⇐ B ∪ {c_i}
15:            zero_rows_last ⇐ zero_rows
16:            super_reduct, reduct, zero_rows ⇐ Evaluate(BM, B)
17:            if zero_rows = zero_rows_last then
18:                B ⇐ B \ {c_i}                  ▷ attribute c_i does not contribute
19:            else
20:                if super_reduct = TRUE then
21:                    if reduct = TRUE then
22:                        RS ⇐ RS ∪ B
23:                    B ⇐ B \ {c_i}
24:                    zero_rows ⇐ zero_rows_last
25:            if i = n − 1 then
26:                k ⇐ LastOne(B)
27:                if k = i then
28:                    B ⇐ B \ {c_k}
29:                    k ⇐ LastOne(B)
30:                if k ≠ j then
31:                    B ⇐ B \ {c_k}
32:                    super_reduct, reduct, zero_rows ⇐ Evaluate(BM, B)
33:                    i ⇐ k + 1
34:                else
35:                    i ⇐ i + 1
36:            else
37:                i ⇐ i + 1
38:    j ⇐ j + 1
```

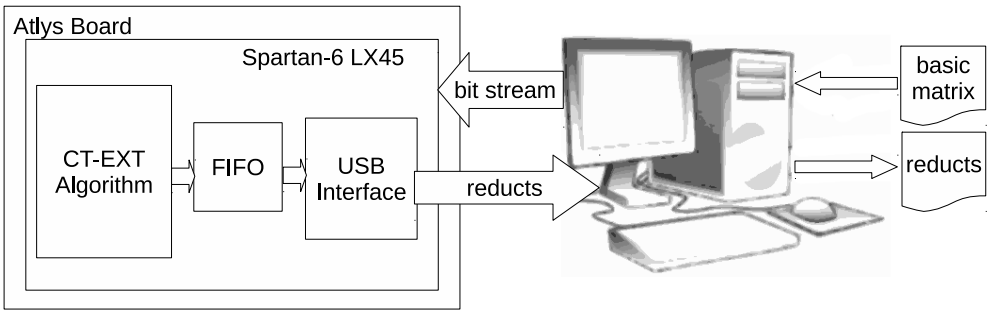

Figure 4.1: Proposed hardware software platform.

4.2.1 Hardware architecture

In the proposed hardware architecture, an attribute subset is handled as an n-tuple, using a positional representation for all the n attributes of a basic matrix BM. Given a subset B, its n-tuple representation has a 1 in the corresponding position i for each $c_i \in B$ and 0 otherwise. The process of deciding whether an n-tuple is a super–reduct of BM involves comparing the candidate against each one of the BM's rows. For software–only implementations, this is a big disadvantage, specially for large matrices with many rows. The proposed hardware architecture exploits the inherent parallelism in the CT–EXT algorithm and evaluates whether a candidate is a reduct, or not, in a single clock cycle. The hardware implementation of CT–EXT is composed of two modules, the BM module and the Candidate Generator module, as shown in Figure 4.2.

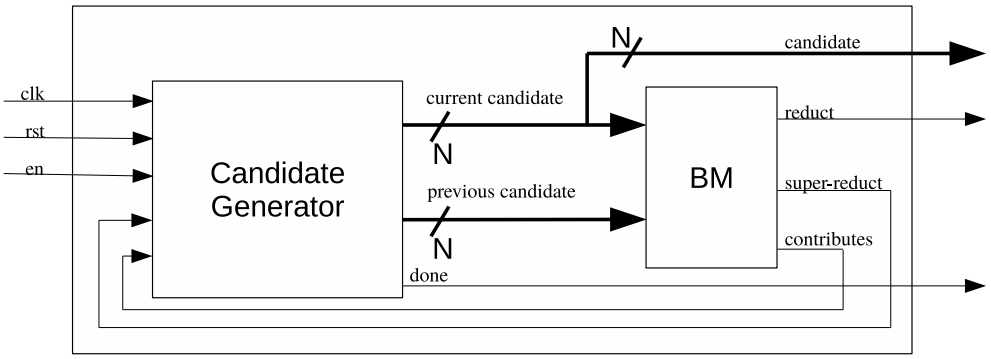

Figure 4.2: CT–EXT Architecture.

The BM module stores the basic matrix and includes all the logic needed to decide whether an n-tuple is a super–reduct. The candidate generator module produces the candidates (n-tuples) to be evaluated by the BM module. In order to calculate the next candidate according to the CT–EXT algorithm, the architecture feedbacks the evaluation result of the previous candidate to the generator module; this drastically reduces the number of candidates tested and consequently the number of iterations needed by the algorithm.

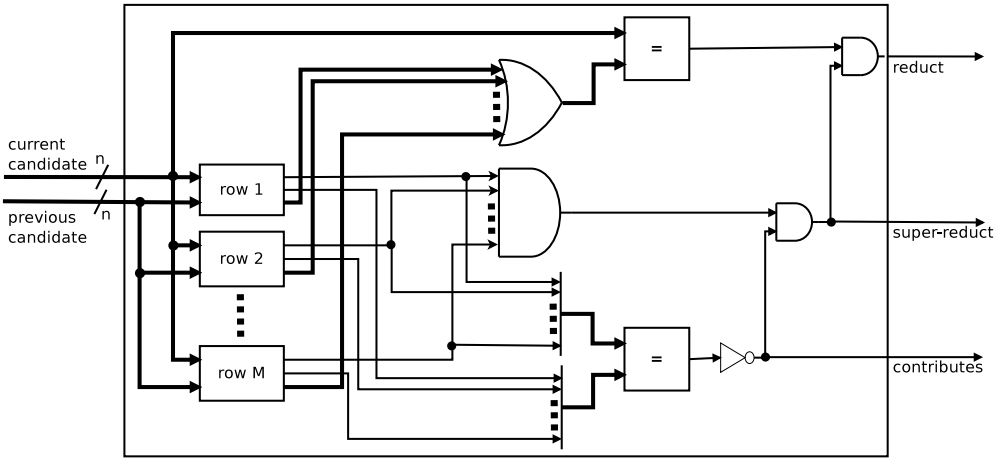

Figure 4.3: BM module.

The BM module is composed of m sub-modules named *row i*, as shown in Figure 4.3. Each *row i* module contains a row (n bits) of the BM matrix and the logic needed to perform a super–reduct evaluation. To decide whether an n-tuple is a super–reduct, a bitwise AND operation is performed between the value stored in each *row i* module and the current candidate, as shown in Figure 4.4. If at least one bit of the AND operation is TRUE, then the output *super–reduct* of that particular *row i* sub-module will be TRUE. The same operation is performed over the previous candidate. If the output *super–reduct* is different from the output *contributes* for any *row i* sub-module, it means that the current candidate reduces the amount of zero rows regarding the

previous candidate and then, the output *contributes* from the BM module becomes TRUE. If the output *super–reduct* of all *row i* sub-modules is TRUE, then the output *super–reduct* of the BM module will be TRUE, which means that the candidate is a super–reduct of BM.

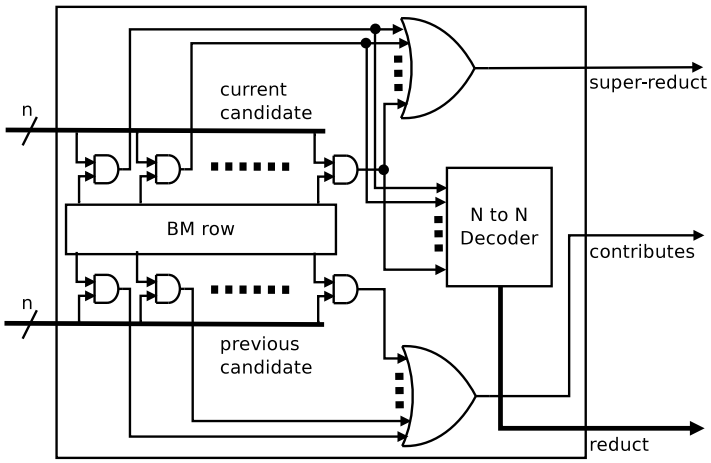

Figure 4.4: BM row.

In order to verify the minimality condition, an *N to N Decoder* receives as input the result of the AND operation between the current candidate and the corresponding BM row. The output from the *N to N Decoder* repeats the input when there is only one bit set to 1, and returns zero otherwise. For those rows with only one bit having a 1 after ANDed with the candidate, the attribute in the position of that bit is indispensable if the candidate is a super–reduct. According to definition of reduct, every attribute must be indispensable.

Taking as example the ordered basic matrix of Table 2.4. In Table 4.2 the irreducibility of $\{c'_0, c'_4, c'_6\}$ is evaluated while the same is done for $\{c'_0, c'_4, c'_5\}$ in Table 4.3. Left rows show the result of the AND operation between each row of BM and the candidate, while those rows in the right show the decoder output taking as input its corresponding left row. In the last row, the result of an OR operation over all above

Table 4.2: Reduct.

Cand. $\{c'_0, c'_4, c'_6\}$							Decoder output						
c'_0	c'_1	c'_2	c'_3	c'_4	c'_5	c'_6	c'_0	c'_1	c'_2	c'_3	c'_4	c'_5	c'_6
1	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	0	0	0	0	0	0	0
Candidate =							1	0	0	0	1	0	1

Table 4.3: No reduct.

Cand. $\{c'_0, c'_4, c'_5\}$							Decoder output						
c'_0	c'_1	c'_2	c'_3	c'_4	c'_5	c'_6	c'_0	c'_1	c'_2	c'_3	c'_4	c'_5	c'_6
1	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0
Candidate ≠							1	0	0	0	1	0	0

bits is shown. According to our previous explanation, the candidate $\{c'_0, c'_4, c'_6\}$ is a reduct given that the result of the OR operation is equal to the candidate itself; while candidate $\{c'_0, c'_4, c'_5\}$ is not.

The candidate generator module (Figure 4.5) uses the feedback from the BM module to calculate the next candidate to be evaluated. The candidate generator module consists of three registers for holding the current candidate ($Curr\_cand$), the previous candidate ($Prev\_cand$) and the last added attribute (J). The values of these registers are updated by the modules EA1, EA2 and A.

Depending on the combination of the input values, the outputs E1A, E2A or A are used for updating the registers. Table 4.4 shows how the registers are updated according to the values of *super–reduct*, *contributes* and J inputs. This operation is computed by the module *sel* shown in Figure 4.5.

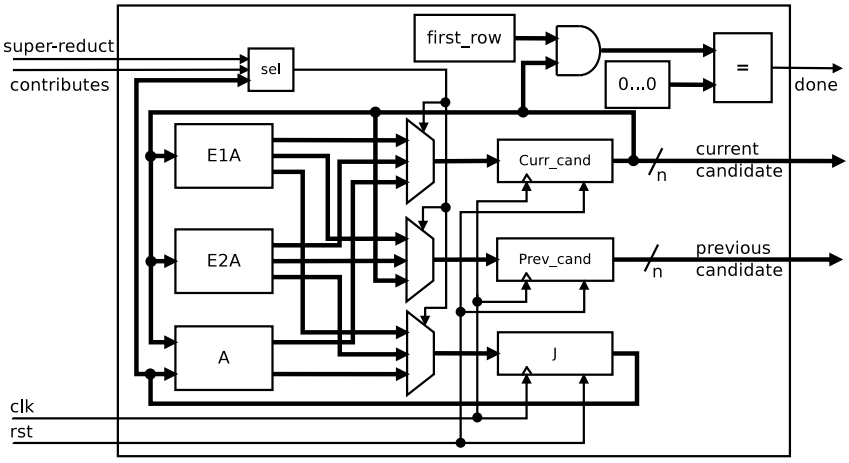

Figure 4.5: Candidate Generator module.

Table 4.4: Candidate Generator Selector.

Priority	Condition	Registers update
1	$J = J_{max}$ (J_{max} = max value of J)	$Curr\_cand \leftarrow$ E2A $Prev\_cand \leftarrow$ E2A $J \leftarrow$ E2A
2	contributes = 0 or super–reduct = 1	$Curr\_cand \leftarrow$ E1A $Prev\_cand \leftarrow$ E1A $J \leftarrow$ E1A
3	contributes = 1 or super–reduct = 0	$Curr\_cand \leftarrow$ A $Prev\_cand \leftarrow Curr\_cand$ $J \leftarrow$ A

The submodule A, shown in Figure 4.6, assigns 1 to the next attribute at the right of the last bit with value 1 in the input candidate. The outputs of the submodule A are the new candidate and $J + 1$.

The submodule $E1A$, shown in Figure 4.7, comprises the Rem_1 (Figure 4.8) and A submodules. The submodule Rem_1 deletes the last attribute added to the input candidate. This action is performed by a priority encoder which locates the last bit with value 1 in the input candidate. Rem_1 outputs represent the previous candidate and the index of the deleted attribute. These outputs are connected to the corresponding inputs of the submodule A, in order to add an attribute in the corresponding position.

Finally, the outputs of $E1A$ represent the new candidate to be evaluated, the previous candidate and the index where the new attribute was added to the current candidate.

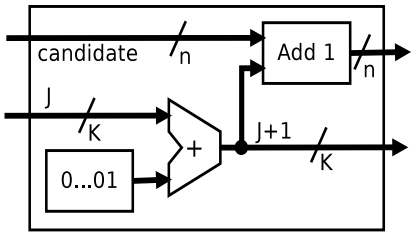

Figure 4.6: Submodule A.

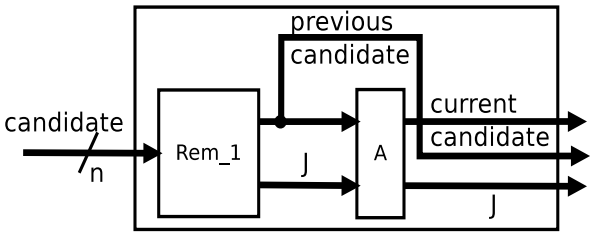

Figure 4.7: Submodule E1A.

Finally, the submodule $E2A$ removes the last two attributes from the input candidate, and then adds the following corresponding attribute. This operation is performed by means of two Rem_1 submodules and an A submodule, as shown in Figure 4.9.

In order to check if the execution of the CT–EXT algorithm has finished, the result of an AND operation between the current candidate and the first row of the basic matrix is compared to the null n-tuple $(0, ..., 0)$, as shown in the upper right corner of Figure 4.5. If the result of this comparison is TRUE, then the output *done* is activated because all further candidates will not satisfy the super–reduct condition over the first row of BM.

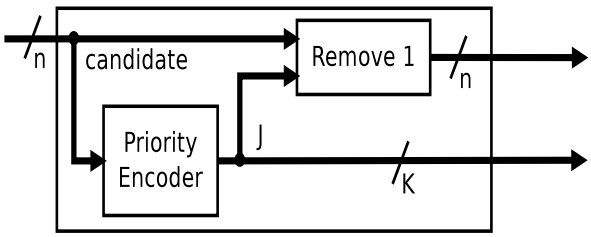

Figure 4.8: Submodule Rem_1.

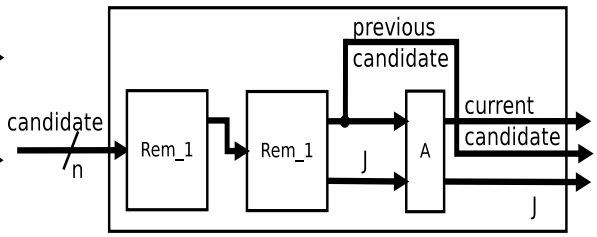

Figure 4.9: Submodule E2A.

The FPGA-based board

The Atlys board from Digilent [11] was selected as the prototyping board. This board is a development and prototyping platform based on a Xilinx Spartan-6 LX45 FPGA, speed grade -3. The Atlys board supports device programming and simplified user-data transfer at a maximum rate of 48MB/s, over a single USB connection.

The communication between the host PC and the FPGA uses the Digilent Synchronous Parallel Interface (DSTM) protocol [10]. Reduct n-tuples, computed by the proposed architecture, are buffered within a FIFO in order to be split into bytes. These bytes are then buffered into a double clocked FIFO [54] to be read from the PC. This last FIFO ensures the output interface operation at 48MHz, as required by the DSTM protocol.

4.2.2 Software Description

The software component allows the user to provide the basic matrix in a plain text file. The software component is responsible for programming the FPGA device and communicating with the board during the reduct computation.

First, the basic matrix is reorganized by setting one of the rows with the minimum amount of ones as the first row and swapping columns in such a way those with a 1 in the first row appear on the left.

Using the arranged basic matrix, a VHDL file is generated and the synthesis and optimization process is started. In this way, the optimization stage takes advantage of the basic matrix data to minimize the FPGA resource utilization. Then, the programming file for the FPGA device is generated.

On the running stage, the software component interacts with the hardware architecture. First, the device is programmed with the bit-file obtained from the previous stage.

Then, the hardware architecture starts computing reducts. The software component keeps pulling through a USB port for new reducts in the output FIFO until the *done* signal is activated in the FPGA.

As a result of the arrangement process, the order of the attributes in the basic matrix is altered as it can be seen in Table 4.1. Consequently, the reducts calculated in the FPGA must be codified according to the order of the columns in the original basic matrix. This task is performed by the software component and then the results are written to the output file.

4.3 Evaluation and Discussion

In order to show the performance of the proposed platform, it was compared against a software implementation of the CT–EXT algorithm [36] and the BT hardware platform previously reported in [30]; which is the most recent hardware implementation for computing reducts reported in the literature.

Either CT–EXT or BT hardware implementations are capable of evaluating a candidate per clock cycle. If both architectures are running at the same frequency, as it will be the case in our experiments, there are two reasons for differences in running time. The first one is the time taken for reorganizing the basic matrix, which is a more complex process in BT, although it can be neglected as it was shown in [34]. The second and the most relevant, is the amount of candidates to be evaluated.

Regarding to the software implementation, the CT–EXT hardware platform has two disadvantages. First, VHDL code is generated for each BM data and a process of synthesis must be accomplished before the algorithm execution; while this is unnecessary in the software version of CT–EXT. Secondly, the software will be running in a PC at a frequency of 3.10GHz while the FPGA architecture will run at 50MHz.

These disadvantages make the hardware approach useful (faster) under two conditions. First, the number of candidates to be evaluated is big enough to overcome the synthesis overhead. Second, the dimensions of the BM are big enough to provide a considerable speedup of the candidate evaluation process. Although the hardware architecture could be designed for a fixed maximum matrix size and receive the BM through the USB port, by doing this, the size of the problem that can be solved would be significantly reduced. The synthesis process comprehend an optimization of the design, taking advantage of the BM data distribution for the reduction of the generated hardware configuration. The number of operations for the evaluation of a single candidate, in the software approach, is proportional to the number of rows and it is directly related to the number of columns in the BM. Using this approach, it is possible to achieve a significant reduction of the processing time, even if operating at a much lower clock frequency, by evaluating a candidate on each clock cycle.

With these points in mind and in order to show the usability of the proposed platform, three kinds of basic matrices were randomly generated. Each type containing different percentage of 1's:

1. Very-low density matrices: approximately 8%.
2. Low density matrices: approximately 33%.
3. Medium density matrices: approximately 45%.

Higher density matrices were discarded because they do not constitute a computationally expensive problem, as stated in [34]. Hereinafter, we will be referring to these three sets of matrices by its approximate density of 1's.

For our experiments, 30 basic matrices of different sizes were randomly generated. A random number generator was used to generate rows, which are filtered for the minimum and maximum number of 1's allowed. In this way the desired density was

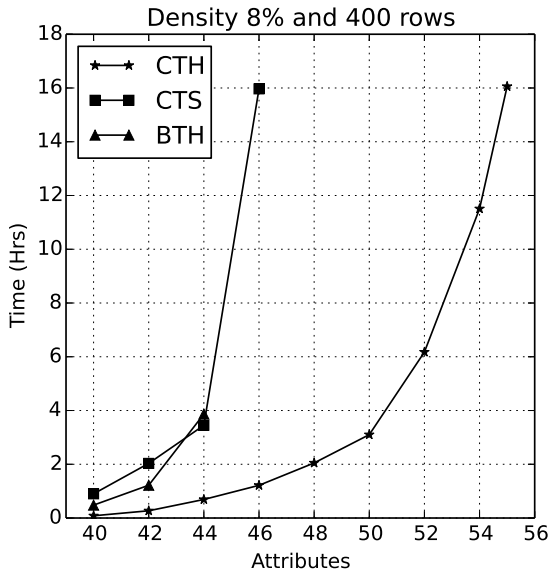

Figure 4.10: Runtime for matrices with a density of 8%.

controlled. If accepted, the row is verified as basic against the saved rows. Basic rows are saved until the desired number of rows is reached.

For the hardware platforms, we measure the runtimes including the time for the following stages: BM input parsing and VHDL code generation, synthesis process, and reduct computation (with the hardware component running at 50MHz). The number of rows for each type of matrices is conditioned by the dimensions of the biggest matrix that may be synthesized at the desired running frequency. All experiments are performed using an Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz for software executions and an Atlys board, powered by a Spartan-6 LX45 FPGA device, for the hardware components. Figures 4.10, 4.11 and 4.12 show graphics of the runtime (in hours) for the three types of basic matrices.

The proposed CT–EXT hardware platform (CTH) results were taken as reference for axis limits in Figures 4.10, 4.11 and 4.12. Slowest executions of the CT–EXT software implementation (CTS) are not shown in order to keep clarity in the figures. The hardware platform for BT (BTH) was not able to met the constrain of 50MHz

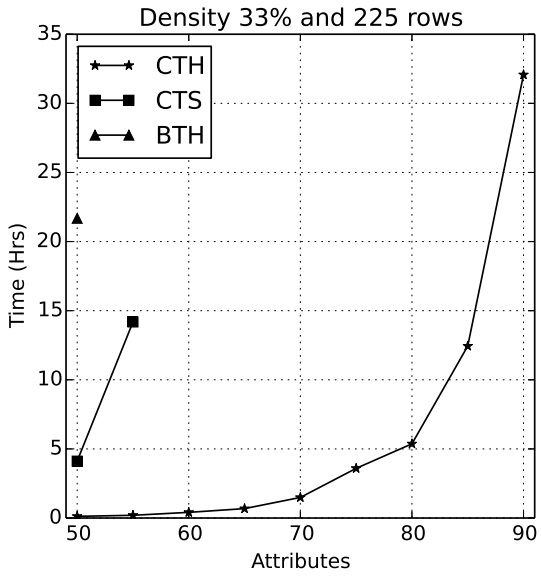

Figure 4.11: Runtime for matrices with a density of 33%.

clock frequency for some matrices and running it at lower frequency resulted in longer runtimes; which fall out of the limits of the figures. We were able to run the three platforms for 4 matrices of different types.

In [34], it was stated that the time for computing reducts does not only depend on the size and density of the BM. This assertion is illustrated by the matrices with 68 and 70 attributes respectively, in Figure 4.12. Although these two matrices have a similar density, the larger matrix requires a shorter runtime.

Table 4.5 shows the runtime for each stage of the data flow, for 400x40, 400x42 and 400x44 very-low density matrices. This table shows that the synthesis time becomes less significant regarding the total time when the problem size increases. From this table, it can also be seen that the main difference between the BT and CT–EXT hardware implementations is the runtime of the reduct computation process. The main reason for this difference is the number of candidates evaluated by each algorithm. This is an explanation about why the BT hardware implementation is slower than the software version of CT–EXT for the largest matrix in Table 4.5.

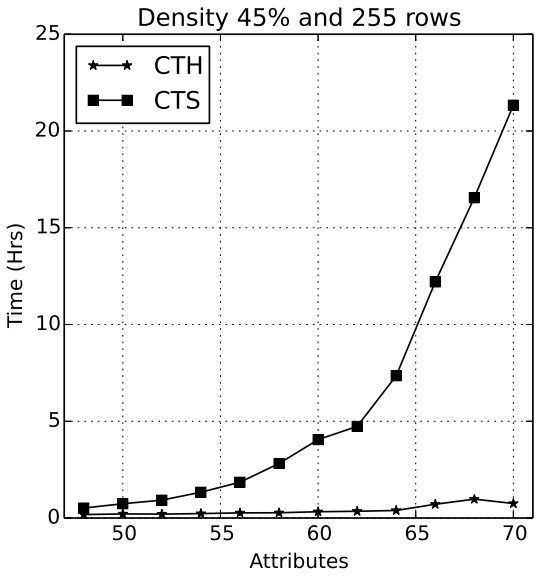

Figure 4.12: Runtime for matrices with a density of 45%.

Tables 4.6 and 4.7 summarize the FPGA resource utilization on our prototyping board. The maximum operation frequency from Tables 4.6 and 4.7 shows that usually the CT–EXT implementation is potentially faster than the modified BT implementation. Resource utilization is directly related to BM dimensions, its density and to a lesser extent to data organization.

As it was shown in these experiments, the proposed hardware platform provides

Table 4.5: Runtime in seconds (broken down for each stage) for 400x40, 400x42 and 400x44 very-low density matrices.

Dimensions	400x40		400x42		400x44	
Stage	CTH	BTH	CTH	BTH	CTH	BTH
Load and file generation	0.05	0.07	0.05	0.06	0.06	0.06
Synthesis process	253	656	401	564	452	612
Algorithm execution	300	1071	970	3826	2031	13311
Total time	554	1727	1372	4390	2484	13924
CTS total time	3238		7320		12420	

Table 4.6: Synthesis summary of resource utilization for a BM with a density of 8% on a Spartan-6 LX45 FPGA device.

Dimensions	400x40		400x44	
Algorithm	BT	CT–EXT	BT	CT–EXT
Slices	1,398 (20%)	983 (14%)	1,554 (22%)	1,209 (17%)
6-input LUTs	4,010 (14%)	2,806 (10%)	4,475 (16%)	3,004 (11%)
Flip-Flops	832 (1%)	852 (1%)	876 (1%)	938 (1%)
Max clock freq	80.44MHz	179.58MHz	84.56MHz	173.24MHz

Table 4.7: Synthesis summary of resource utilization for a BM with a density of 33% on a Spartan-6 LX45 FPGA device.

Dimensions	225x50		225x55	
Algorithm	BT	CT–EXT	BT	CT–EXT
Slices	1,381 (20%)	1,554 (22%)	1,455 (21%)	1,562 (22%)
6-input LUTs	3,769 (13%)	4,315 (15%)	4,135 (15%)	5,026 (18%)
Flip-Flops	949 (1%)	980 (1%)	1,002 (1%)	1,039 (1%)
Max clock freq	87.46MHz	155.40MHz	85.27MHz	156.35MHz

higher processing performance than the software implementation of the CT–EXT algorithm for the used matrices. This behavior is possible because the hardware component of the proposed platform is capable of testing whether a candidate is a super–reduct of a BM in a single clock cycle, independently of the number of columns and rows, whereas the software implementation runtime will significantly increase for matrices with a large number of rows.

Experimental results show that the proposed platform beats the software implementation of the CT–EXT algorithm, with ratios of around **one order** of magnitude. However, for large enough datasets this improvement could be significantly higher, as it can be inferred from Figure 4.12.

4.4 Concluding Remarks

In this chapter, we presented the design and implementation of a new hardware software platform for computing all reducts in a dataset. Unlike most of the existing hardware architectures for feature selection, our proposal computes all the minimal subsets of attributes that preserve the discernibility capacity of the original feature set. The good performance of our hardware implementation, compared to the software approach, is feasible due to the high level of parallelism implicit in the candidate evaluation process of the CT–EXT algorithm; which can be efficiently implemented on an FPGA. This proposed architecture offers an alternative to a previous hardware implementation; being faster in most of the cases, by evaluating less candidates.

Our experiments also showed that the proposed platform uses fewer hardware resources and it is able to run at higher clock frequency than the hardware implementation of BT. This characteristic allows processing larger matrices, since the maximum size of the problem that can be solved in a hardware architecture is conditioned by its resource requirements.

Even though our platform can process larger basic matrices than the BT implementation, its resource utilization determines the maximum size of the basic matrix that can be solved (this is, indeed, the main limitation of this proposal). In this way, hardware platforms for reduct computation are faster for basic matrices big enough to overcome the delay of the synthesis process and to take advantage of the single clock candidate evaluation; but the size of these basic matrices is limited by the FPGA resources. This issue of hardware platforms drastically reduces their practical application.

With this precedent, we directed our PhD research to the development of new algorithms for reducing the runtime of reduct computation. The algorithms proposed through this research can be further implemented in a hardware fashion; but due to

the current limits in FPGA resources, these implementations are beyond our goals.

Chapter 5

A New Algorithm for Computing All Reducts

As it was shown in Chapter 3, several algorithms have been reported to reduce the cost of reduct computation. Unfortunately, most of these algorithms relay on high cost operations for candidate evaluation. Therefore, in this chapter, we propose a new algorithm, GCreduct, for computing all reducts of a decision system, based on the pruning properties of *Gap elimination* and *attribute Contribution*. Our proposed algorithm uses simpler operations for candidate evaluation, which allows reducing the runtime required for computing all reducts. In addition, an experimental study for finding a relation between some properties of the basic matrix and the fastest algorithms for reduct computation, is presented.

The contents in this chapter are structured as follows. In Section 5.1, we introduce the GCreduct algorithm for computing all reducts of a decision system. An evaluation of the proposed algorithm and a discussion of the experimental results are presented in Section 5.2. Finally, Section 5.3 shows our concluding remarks.

5.1 GCreduct

In this section, we introduce the GCreduct algorithm for computing all reducts of a decision system. In Subsection 5.1.1, we present the pruning properties used in GCreduct. Then, in Subsection 5.1.2, we introduce the GCreduct algorithm; illustrating its execution over the decision system of Table 2.1.

5.1.1 Pruning Properties for GCreduct

The concept of gap was first introduced for LEX [39], to avoid the evaluation of candidates that are subsets of reducts or non super–reducts.

Definition 5.1 *Let $L = [c_{i_0}, ..., c_{i_s}]$ be an ordered list of attributes. If there exists an attribute $c_{i_p} \in L$ such that $i_p = \max\{i_q | i_{q+1} \neq i_q + 1\}; 0 \leq q < s$, c_{i_p} is the gap of L.*

In other words, the gap of L is the attribute with the highest index satisfying that its consecutive attribute in L is not its consecutive attribute (column) in the basic matrix. Notice, from the notation used above, that i_q and i_{q+1} are consecutive indexes in L while i_q and $i_q + 1$ are consecutive indexes in the basic matrix.

Let us take, for example, a basic matrix with 7 attributes $(c_0 - c_6)$, then:

$$\begin{aligned} L_1 &= [c_0, c_1, c_2, c_3] && \text{there is no gap} \\ L_2 &= [c_0, c_1, c_2, c_5, c_6] && \text{the gap is } c_2 \\ L_3 &= [c_0, c_1, c_2, c_4, c_6] && \text{the gap is } c_4 \end{aligned}$$

In the first example, there is no gap since all the attributes in L_1 are consecutive in the basic matrix. In the second example, the consecutive attribute of c_2 in L_2 is c_5, which is not its consecutive attribute in the basic matrix. Thus, for L_2, the gap is c_2. This is also the case for c_2 in L_3, however, since c_4 has a higher index and its consecutive attribute in L_3 is not c_5, c_4 is the gap for L_3.

The pruning property of gap elimination is supported by the proposition 5.1. The proof of this proposition can be seen in [32].

Proposition 5.1 *Let BM be a basic matrix and $L = [c_{i_0}, ..., c_{i_s}]$ be an ordered list associated to a reduct, such that c_{i_s} is the last attribute in BM. If there is a gap c_{i_p} in L, there is no list Y associated to a reduct, such that $L \prec Y \prec L'$ and $L \neq Y \neq L'$, where $L' = [c_{i_0}, \ldots, c_{i_{p-1}}, c_{i_p+1}]$.*

From Proposition 5.1 we have the following corollary; which is relevant for the proposed algorithm in order to avoid unnecessary evaluations.

Corollary 5.1 *Let BM be a basic matrix and* $L = [c_{i_0}, ..., c_{i_s}]$ *be an ordered list associated to a non super–reduct, such that* c_{i_s} *is the last attribute in BM. If there is a gap* c_{i_p} *in L, there is no list Y associated to a reduct, such that* $L \prec Y \prec L'$ *and* $Y \neq L'$*, where* $L' = [c_{i_0}, ..., c_{i_{p-1}}, c_{i_p+1}]$.

Using Proposition 5.1 and Corollary 5.1, all candidates between L and L', in the lexicographical order, can be discarded.

For a fast evaluation of candidates, in GCreduct, the columns of a basic matrix BM are coded as binary words with as many bits as rows in BM. The cumulative mask for an attribute c_i, denoted as cm_{c_i}, is defined as the binary word representing the i-th column in BM. The cumulative mask for an ordered list of attributes $B = [c_{i_1}, c_{i_2}, ..., c_{i_k}]$ is defined as $cm_B = cm_{c_{i_1}} \vee cm_{c_{i_2}} \vee ... \vee cm_{c_{i_k}}$ where $\vee$ represents the binary OR operator. It is not hard to see that the number of 0's in cm_B is the same as the number of zero rows in the sub-matrix of the basic matrix, considering only the attributes in B. According to Definition 4.2, c_i contributes to B iff $cm_{B+[c_i]}$ has more 1's than cm_B. The value of $cm_{B+[c_i]}$ can be computed incrementally, as in fast–CT–EXT [38] and fast–BR [24], due to the associative property of the OR operation (we can compute $cm_{B+[c_i]} = cm_B \vee cm_{c_i}$). Notice from this last formulation, that c_i contributes to B iff $cm_{B+[c_i]} \neq cm_B$, since $cm_{B+[c_i]}$ cannot have less 1's than cm_B. It is easy to see, from Definition 4.1, that $B \subseteq C$ is a super–reduct iff $cm_B = (1, ..., 1)$ (cm_B has a 1 in every bit).

In order to determine whether a super–reduct is a reduct (verifying the minimality condition) the exclusion mask, introduced in [23], plays a fundamental role.

Definition 5.2 *Let BM be a basic matrix and B be an ordered list of attributes. We call exclusion mask of B, denoted as em_B, to the binary word in which the $i-th$ bit is 1 if the $i-th$ row in BM has a 1 in only one column of those columns corresponding to attributes in B, and it is 0 otherwise.*

For instance, from the basic matrix of Table 2.4 we have:

$$\begin{aligned} em_{[c_0,c_1,c_2]} &= (0,1,1,0,1,1) \\ em_{[c_0,c_1,c_2,c_3]} &= (0,1,0,1,1,0) \\ em_{[c_0,c_1,c_2,c_3,c_4]} &= (1,0,0,1,1,0) \end{aligned}$$

Fast–BR [24], introduced the following two propositions to support the cumulative computation of the exclusion mask.

Proposition 5.2 *Let B be an ordered list of attributes and $c_i \in C$ be an attribute, such that $c_i \notin B$. The exclusion mask of $B + [c_i]$ is computed as follows:*

$$em_{B+[c_i]} = (em_B \wedge \neg cm_{c_i}) \vee (\neg cm_B \wedge cm_{c_i})$$

where cm refers to the cumulative mask.

Proposition 5.3 *Let B be an ordered list of attributes and $c_i \in C$ be an attribute, such that $c_i \notin B$. If $\exists c_x \in B$ such that $em_{B+[c_i]} \wedge cm_{c_x} = (0,...,0)$. Then, $B + [c_i]$ cannot be a subset of any reduct, and we say that c_i is exclusionary with B.*

We call exclusion evaluation to the application of Proposition 5.3. This proposition allows us to discard the supersets of $B + [c_i]$ if c_i is exclusionary with B.

5.1.2 The GCreduct algorithm

GCreduct finds all reducts in the basic matrix of a decision system. This algorithm explores the search space evaluating some candidates and discarding others based on previous evaluations. Unlike other works [51, 24] where candidate evaluation is performed through operations with high cost, GCreduct uses a simpler candidate evaluation based on gap elimination and attribute contribution to reduce the runtime.

The GCreduct algorithm traverses the search space following the lexicographical order. Once a new attribute is added, it is removed if it does not reduce the number of zero rows regarding the previous candidate (i.e. the attribute does not contribute). If the new attribute contributes, the candidate is evaluated for the super–reduct and reduct conditions. If the candidate is a reduct, it is saved in the result set. If the current candidate is a super–reduct, the new attribute is removed, since it does not make sense to evaluate supersets of a super–reduct. This process continues until the last attribute from the basic matrix has been included in the candidate. At this point, the algorithm searches for a gap in the candidate to avoid unneeded evaluations. The algorithm continues until the first attribute in the candidate has a zero in the first row of the basic matrix. Once this condition is reached, the algorithm finishes because there are no remaining reducts (Proposition 4.2).

The pseudocode for GCreduct is shown in Algorithm 5.1. The algorithm operates over the arranged basic matrix, and as a preprocessing stage, superfluous attributes [21] are removed from the basic matrix. After initializing the current candidate with the first attribute, the cumulative mask is updated. Then, the attribute contribution is evaluated using Definition 4.2. For those candidate subsets with a contributing attribute, the super–reduct condition is evaluated using Definition 4.1. For each detected super–reduct, Proposition 5.3 is evaluated in order to determine whether a super–reduct is a reduct, to be saved in the result (RR). At this point, the candidate evaluation is

finished and the *candidateGenerator* procedure is called to evaluate a new candidate subset.

Algorithm 5.1 GCreduct algorithm for computing all reducts

Input: BM ▷ The arranged basic matrix
Output: RR ▷ The set of all reducts
1: $RR \Leftarrow \emptyset$
2: $B \Leftarrow \emptyset$ ▷ Subset of attributes in the candidate
3: $c \Leftarrow 0$ ▷ New attribute to add in the candidate
4: **while not** done **do**
5: $reduct \Leftarrow$ False, $superReduct \Leftarrow$ False, $contributes \Leftarrow$ False
6: $cm_{B+[c]} \Leftarrow$ updateCM$(B + [c])$
7: **if** $cm_{B+[c]} \neq cm_B$ **then**
8: $contributes \Leftarrow$ True
9: **if** $cm_{B+[c]} = (1, ..., 1)$ **then**
10: $superReduct \Leftarrow$ True
11: $reduct \Leftarrow$exclusion$(B + [c])$
12: **if** $reduct$ **then**
13: $RR \Leftarrow RR \cup \{B + [c]\}$
14: $B + [c], done \Leftarrow$candidateGenerator$(B + [c], contributes, superReduct, reduct)$

In Algorithm 5.2, the pseudocode for updating the cumulative mask of a candidate $B + [c]$ is shown. Cumulative masks are stored in the CM array, indexed by the last attribute in B for further computations. The function $getLast(B)$ returns the last attribute in B. The time complexity of this procedure is $\Theta(m)$, where m is the number of rows in the basic matrix, because of the bit wise operation of line 5.

Algorithm 5.2 *updateCM* procedure

Input: $B + [c]$ ▷ Current candidate
Output: $cm_{B+[c]}$ ▷ Updated cumulative mask
1: **if** $B = \emptyset$ **then**
2: $cm_B \Leftarrow (0, ..., 0)$
3: **else**
4: $cm_B \Leftarrow CM[$getLast$(B)]$
5: $cm_{B+[c]} \Leftarrow cm_B \vee cm_c$
6: $CM[c] \Leftarrow cm_{B+[c]}$

The pseudocode for the gap elimination is shown in Algorithm 5.3. Every consecutive attribute in the current candidate, starting from the last attribute is eliminated as it was explained in Definition 5.1. The time complexity of this procedure is $\Theta(n)$,

Algorithm 5.3 *eliminateGAP* procedure

Input: B ▷ Ordered list of attributes
Output: B ▷ Ordered list of attributes after the gap elimination

```
1: last ⇐ LastAttribute
2: while getLast(B) = (last − 1) do
3:     last ⇐ getLast(B)
4:     B ⇐ B \ last
5:     if |B| = 1 then
6:         break
```

where n is the number of columns in the basic matrix; this complexity is determined by the number of iterations in the while loop. Since all other instructions within the *candidateGenerator* procedure are $\Theta(1)$, we can conclude that the overall time complexity of *candidateGenerator* is $\Theta(n)$. In the same way, it can be seen that the overall time complexity of a candidate evaluation in GCreduct, is $\Theta(nm)$. However, since computing all reducts is an NP–hard problem, the number of candidate evaluations has an exponential relation to n.

The pseudocode for the exclusion evaluation is shown in Algorithm 5.4. First, the exclusion mask is cumulatively computed using Proposition 5.2. Then, every attribute in B is evaluated for exclusion applying Proposition 5.3. The time complexity of this procedure is $\Theta(nm)$ because there are $\Theta(n)$ loops iterations over the bitwise operations with complexity $\Theta(m)$.

Algorithm 5.4 *exclusion* procedure

Input: $B + [c]$ ▷ Current candidate
Output: *reduct* ▷ Boolean variable indicating whether the current candidate is a reduct

```
1: em ⇐ (0, ..., 0), cm ⇐ (0, ..., 0)
2: for all x ∈ B + [c] do ▷ Computing exclusion mask (Proposition 5.2)
3:     em ⇐ (em ∧ ¬cm_x) ∨ (¬cm ∧ cm_x)
4:     cm ⇐ cm ∨ cm_x
5: reduct ⇐ True
6: for all x ∈ B do ▷ Exclusion evaluation (Proposition 5.3)
7:     if em ∧ cm_x = (0, ..., 0) then
8:         reduct ⇐ False
9:         break
```

In Algorithm 5.5, the pseudocode of the procedure for generating the next candidate

$(B + [c])$ is shown. The function $Next$ returns the following attribute in the arranged basic matrix. $LastAttribute$ holds the position of the last attribute in the basic matrix. If the last attribute of the basic matrix has already been included in the current candidate, the following actions are taken. If the current candidate is a reduct (Proposition 5.1) or it is not a super–reduct (Corollary 5.1), the gap is eliminated. Otherwise, the lexicographical order is followed.

If the last attribute is not reached yet, there are two possibilities:

1. The current candidate is a super–reduct or the current attribute c does not contribute to B; then c is replaced by the next attribute in the arranged basic matrix. In this way, all the supersets of $B + [c]$ are pruned, taking advantage of the condition 2 of Definition 2.1 (if the current candidate is a super–reduct); or applying Proposition 4.1 (if c does not contribute to B).

2. The attribute c contributes to B and the current candidate is not a super–reduct; then the current attribute is added to B and the next attribute in the basic matrix is loaded to c; following the lexicographical order.

Each time a new candidate with a single attribute $(B + [c] = [c])$ is generated, the column of the basic matrix corresponding to c is verified to detect a zero in its first row. If it is the case, the algorithm finishes. This final pruning is possible because of Proposition 4.2.

In GCreduct, we compute the exclusion mask and evaluate the attribute exclusion, using Proposition 5.3, for those candidates detected as super–reducts in order to verify whether they are reducts. super–reducts are, most of the times, a small fraction of the evaluated candidates. On the other hand, in algorithms such as fast–BR [24] and RGonCRS [51], for each candidate with a contributing attribute, the exclusion evaluation is performed. In this way, fast–BR and RGonCRS avoid the evaluation of all the

Algorithm 5.5 *candidateGenerator* procedure

Input: $B + [c]$ ▷ Current candidate
contributes ▷ Boolean variable indicating whether the new attribute contributes
superReduct ▷ Boolean variable indicating whether the current candidate is a super–reduct
reduct ▷ Boolean variable indicating whether the current candidate is a reduct
Output: $B + [c]$ ▷ Next candidate
done ▷ Boolean variable indicating whether the condition to finish have been reached

```
1: done ⇐ False
2: if c = LastAttribute then                          ▷ Last column of the basic matrix is reached
3:     if reduct or not superReduct then              ▷ Eliminate the gap
4:         eliminateGAP(B)
5:     c ⇐ Next(getLast(B))
6:     B ⇐ B \ getLast(B)
7: else
8:     if not contributes or superReduct then
9:         c ⇐ Next(c)                                ▷ Eliminate c and add the next attribute
10:    else
11:        B ⇐ B + [c]                                ▷ Include c
12:        c ⇐ Next(c)                                ▷ Add the next attribute
13: if B = ∅ and cm_c[0] = 0 then                     ▷ The current attribute c has a 0 in the first row
14:    done ⇐ True
```

supersets of a candidate which has an exclusionary attribute. However, the exclusion evaluation is the procedure with the highest complexity into these algorithms ($\Theta(nm)$), and it is evaluated for most candidates by fast–BR and RGonCRS.

In Table 5.1, we show an example of the execution of GCreduct over the basic matrix of Table 4.1. The first column numerates the candidate evaluations. The second column shows the position of the current candidate, according to the lexicographical order; while the third column shows the evaluated candidates. Notice that, for this example, there are 127 attribute subsets in the search space from which, only 39 have to be analyzed. In the last column, some comments about the evaluation of the current candidate and the generation of the next one, are included.

Table 5.1: Execution of GCreduct over the basic matrix shown in Table 4.1.

Iter	Pos	Candidate	Comments
1	1	$[c'_0]$	c'_0 contributes to [ ] but the candidate is not a super–reduct. Add a new attribute.
2	2	$[c'_0, c'_1]$	c'_1 does not contributes to $[c'_0]$. Remove c'_1.
3	34	$[c'_0, c'_2]$	c'_2 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
4	35	$[c'_0, c'_2, c'_3]$	c'_3 contributes to $[c'_0, c'_2]$ but the candidate is not a super–reduct. Add a new attribute.
5	36	$[c'_0, c'_2, c'_3, c'_4]$	The candidate is a super–reduct but it is not a reduct. Remove c'_4.
6	40	$[\mathbf{c'_0}, \mathbf{c'_2}, \mathbf{c'_3}, \mathbf{c'_5}]$	The candidate is a reduct and it is saved. Remove c'_5.
7	42	$[c'_0, c'_2, c'_3, c'_6]$	The candidate is a super–reduct but it is not a reduct. The lexicographical order is followed.
8	43	$[\mathbf{c'_0}, \mathbf{c'_2}, \mathbf{c'_4}]$	The candidate is a reduct and it is saved. Remove c'_4.
9	47	$[c'_0, c'_2, c'_5]$	c'_5 contributes to $[c'_0, c'_2]$ but the candidate is not a super–reduct. Add a new attribute.
10	48	$[c'_0, c'_2, c'_5, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_2) is eliminated.
11	50	$[c'_0, c'_3]$	c'_3 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
12	51	$[c'_0, c'_3, c'_4]$	c'_4 contributes to $[c'_0, c'_3]$ but the candidate is not a super–reduct. Add a new attribute.
13	52	$[c'_0, c'_3, c'_4, c'_5]$	c'_5 does not contributes to $[c'_0, c'_3, c'_4]$. Remove c'_5.
14	54	$[c'_0, c'_3, c'_4, c'_6]$	The candidate is a super–reduct but it is not a reduct. The lexicographical order is followed.
15	55	$[c'_0, c'_3, c'_5]$	c'_5 contributes to $[c'_0, c'_3]$ but the candidate is not a super–reduct. Add a new attribute.
16	56	$[c'_0, c'_3, c'_5, c'_6]$	The candidate is a super–reduct but it is not a reduct. The lexicographical order is followed.
17	57	$[\mathbf{c'_0}, \mathbf{c'_3}, \mathbf{c'_6}]$	The candidate is a reduct and it is saved. Since the last attribute is included, the gap (c'_3) is eliminated.

Iter	Pos	Candidate	Comments
18	58	$[c'_0, c'_4]$	c'_4 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
19	59	$[c'_0, c'_4, c'_5]$	c'_5 does not contributes to $[c'_0, c'_4]$. Remove c'_5.
20	61	$[\mathbf{c'_0}, \mathbf{c'_4}, \mathbf{c'_6}]$	The candidate is a reduct and it is saved. Since the last attribute is included, the gap (c'_4) is eliminated.
21	62	$[c'_0, c'_5]$	c'_5 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
22	63	$[c'_0, c'_5, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_0) is eliminated.
23	65	$[c'_1]$	c'_1 contributes to [ ] but the candidate is not a super–reduct. Add a new attribute.
24	66	$[c'_1, c'_2]$	c'_2 contributes to $[c'_1]$ but the candidate is not a super–reduct. Add a new attribute.
25	67	$[c'_1, c'_2, c'_3]$	c'_3 contributes to $[c'_1, c'_2]$ but the candidate is not a super–reduct. Add a new attribute.
26	68	$[c'_1, c'_2, c'_3, c'_4]$	c'_4 contributes to $[c'_1, c'_2, c'_3]$ but the candidate is not a super–reduct. Add a new attribute.
27	69	$[c'_1, c'_2, c'_3, c'_4, c'_5]$	The candidate is a super–reduct but it is not a reduct. Remove c'_5.
28	71	$[c'_1, c'_2, c'_3, c'_4, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_4) is eliminated.
29	72	$[\mathbf{c'_1}, \mathbf{c'_2}, \mathbf{c'_3}, \mathbf{c'_5}]$	The candidate is a reduct and it is saved. Remove c'_5.
30	74	$[c'_1, c'_2, c'_3, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_3) is eliminated.
31	75	$[c'_1, c'_2, c'_4]$	c'_4 contributes to $[c'_1, c'_2]$ but the candidate is not a super–reduct. Add a new attribute.
32	76	$[\mathbf{c'_1}, \mathbf{c'_2}, \mathbf{c'_4}, \mathbf{c'_5}]$	The candidate is a reduct and it is saved. Remove c'_5.
33	78	$[c'_1, c'_2, c'_4, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_4) is eliminated.
34	79	$[c'_1, c'_2, c'_5]$	c'_5 contributes to $[c'_1, c'_2]$ but the candidate is not a super–reduct. Add a new attribute.
35	80	$[c'_1, c'_2, c'_5, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_2) is eliminated.
36	82	$[c'_1, c'_3]$	c'_3 contributes to $[c'_1]$ but the candidate is not a super–reduct. Add a new attribute.

Iter	Pos	Candidate	Comments
37	83	$[c'_1, c'_3, c'_4]$	c'_4 contributes to $[c'_1, c'_3]$ but the candidate is not a super–reduct. Add a new attribute.
38	84	$[c'_1, c'_3, c'_4, c'_5]$	c'_5 contributes to $[c'_1, c'_3, c'_4]$ but the candidate is not a super–reduct. Add a new attribute.
39	85	$[c'_1, c'_3, c'_4, c'_5, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_1) is eliminated.
40	97	$[c'_2]$	The algorithm finishes because the column of the basic matrix corresponding to the leftmost attribute in the candidate (c'_2) has a zero in the first row. $RR = \{[c'_0, c'_2, c'_3, c'_5], [c'_0, c'_2, c'_4], [c'_0, c'_3, c'_6], [c'_0, c'_4, c'_6], [c'_1, c'_2, c'_3, c'_5], [c'_1, c'_2, c'_4, c'_5]\}$

In this example, the algorithm starts with the list $B = [\,]$ and its cumulative mask $cm_B = (000000)$. The first candidate is built by assigning to c the current attribute c'_0 ($B + [c] = [c'_0]$) and we have $cm_{[c'_0]} = (111000)$. According to Definition 4.2, this attribute contributes, thus it is added to $B = [c'_0]$. Following, the next attribute is added by doing $c = c'_1$. The new cumulative mask is $cm_{[c'_0,c'_1]} = (111000) = cm_{[c'_0]}$ and c'_1 does not contribute to B. At this point, all the supersets of $[c'_0, c'_1]$ are pruned according to Proposition 4.1. Thus, $B = [c'_0]$ and $c = c'_2$, and the candidates from the third to the 33rd, in the lexicographical order, are pruned as it can be seen in Table 5.1. This time, we have $cm_{[c'_0,c'_2]} = (111100) \neq cm_{[c'_0]}$ so that c'_2 contributes to $[c'_0]$ and it is added to $B = [c'_0, c'_2]$ while $c = c'_3$. Again c'_3 contributes to $[c'_0, c'_2]$ and $cm_{[c'_0,c'_2,c'_3]} = (111110) \neq cm_{[c'_0,c'_2]}$. Now, the candidate is $B = [c'_0, c'_2, c'_3]$ and $c = c'_4$ with $cm_{[c'_0,c'_2,c'_3,c'_4]} = (111111)$; which is a super–reduct. The next step is the exclusion evaluation over this super–reduct to verify whether it is a reduct.

The exclusion verification starts with $cm_B = em_B = (000000)$. The final value $em_{[c'_0,c'_2,c'_3,c'_4]} = (101101)$ is computed as it is illustrated in Table 5.2, by using Proposition 5.2 (Algorithm 5.4). Since $cm_{c'_3} \wedge (101101) = (000000)$, we have that $[c'_0, c'_2, c'_3, c'_4]$ is not a reduct. In order to prune supersets of this candidate, the last attribute (c'_4) is removed and we have $B = [c'_0, c'_2, c'_3]$ with $c = c'_5$ for the next evaluation. Now,

the candidate $[c'_0, c'_2, c'_3, c'_5]$ is a reduct with $em_{[c'_0,c'_2,c'_3,c'_5]} = (100111)$. The rest of the example in Table 5.1 follows this process. The algorithm finishes after the candidate $[c'_1, c'_3, c'_4, c'_5, c'_6]$ while evaluating $[c'_2]$, which has a 0 in the first row of the basic matrix.

Table 5.2: Exclusion mask computation for $[c'_0, c'_2, c'_3, c'_4]$.

$B + [c]$	cm_c	cm_B	em_B	$em_{B+[c]}$
$[c'_0]$	(111000)	(000000)	(000000)	(111000)
$[c'_0, c'_2]$	(010100)	(111000)	(111000)	(101100)
$[c'_0, c'_2, c'_3]$	(000010)	(010100)	(101100)	(101110)
$[c'_0, c'_2, c'_3, c'_4]$	(000011)	(000010)	(101110)	(101101)

During the search process, GCreduct evaluates some attribute subsets while discards some others. Propositions 5.1, 4.1 and 4.2 as well as the minimality condition of reducts, guarantee that none of the discarded attribute subsets is a reduct. This is summarized in the following proposition.

Proposition 5.4 *The algorithm GCreduct computes all reducts of a decision system.*

Proof. *The algorithm GCreduct traverses the search space of attribute subsets in the lexicographical order. During the search process, some attribute subsets are evaluated while other ones are discarded in the following cases:*

- *Supersets of an attribute subset which has a non contributing attribute (Proposition 4.1).*
- *Supersets of a super–reduct (which cannot be a reduct based on the condition 2 of the subsection 2.2).*
- *Subsets of a reduct (Proposition 5.1), or a non super–reduct (Corollary 5.1), which includes the last attribute of the basic matrix.*
- *The remaining candidates, when the column of the basic matrix corresponding to the leftmost attribute in the candidate has a zero in the first row (Proposition 4.2).*

Since all attribute subsets discarded by the searching process are certainly not reducts, we can state that GCreduct computes all reducts of a decision system.

5.2 Evaluation and Discussion

In this section, we perform a comparative analysis of the proposed algorithm (GCreduct) versus RGonCRS, fast–CT–EXT, and fast–BR. These are the most recent and fastest algorithms for reduct computation reported in the literature. Since all these algorithms have exponential complexity, then we perform an experimental comparison among them. We evaluate all algorithms over decision systems from the UCI machine learning repository [3] and synthetic basic matrices. In all cases the author's implementation of the algorithm was used. All experiments were run on a Core i7-5820K Intel processor at 3.30GHz, with 32GB in RAM, running GNU/Linux.

Since RGonCRS is implemented in Matlab and it operates directly over the decision system, in Subsection 5.2.1, we compare GCreduct against RGonCRS over decision systems by using a Matlab implementation of the proposed algorithm. Then, in Subsection 5.2.2, we show a comparative study of GCreduct against fast–CT–EXT and fast–BR over basic matrices computed from decision systems taken from the UCI repository, by using a Java implementation of the proposed algorithm. Finally, in Subsection 5.2.3, we present an experimental study including fast–CT–EXT, GCreduct and fast–BR over synthetic basic matrices. In this final experiment, we discuss the algorithms' performance regarding some properties of the basic matrix.

5.2.1 GCreduct vs. RGonCRS

In order to make a fair comparison against RGonCRS, we used its author's implementation in Matlab and we implemented GCreduct in Matlab as well. For this experiment,

we selected 15 decision systems from the UCI machine learning repository [3]. Objects with missing data were removed as in [51]. For numerical attributes, we used the Weka's equal width discretization filter with 10 bins, as describe in [12]. This experiment was run using Matlab R2015a.

Table 5.3: RGonCRS and GCreduct runtime for decision systems taken from the UCI repository.

Decision system	Attributes	Instances	Reducts	RGonCRS runtime(s)	GCreduct runtime(s)
Chess (kr-vs-kp)	37	3196	4	124.31	**4.79**
Connect-4	43	6756	35	**5245.23**	26432.14
Credit-g	21	1000	846	23.95	**4.78**
Cylinder-bands	40	512	23534	152.19	**148.50**
Dermatology	35	366	112708	12683.77	**2359.98**
Diabetes	50	101766	77349	**2507.16**	10261.38
Flags	30	194	23543	**316.50**	520.41
Keyword-activity	37	1530	3	**3.82**	250.38
Landsat (train)	37	4435	12412798	>1500000	**603970.39**
Lung-cancer	57	32	4183355	**31548.48**	47239.18
QSAR-biodeg	42	1055	256	884.79	**57.00**
Sponge	46	76	10992	**68.23**	144.36
Student-mat	32	395	679121	21632.40	**5153.27**
Student-por	32	649	851584	27568.81	**11852.47**
Waveform	22	5000	86977	8739.26	**5324.35**

Table 5.3 shows the runtime of both algorithms for each decision system. The runtime for GCreduct, shown in the last column of Table 5.3, includes the time taken for computing the basic matrix. Notice that GCreduct performed faster than RGonCRS in most decision systems.

5.2.2 GCreduct vs. fast–CT–EXT and fast–BR

For this experiment, we implemented the proposed algorithm in Java to make a fair comparison against the Java implementations of fast–BR and fast–CT–EXT, provided by their authors. The same 15 decision systems of the previous experiment were used. The runtime in seconds for the three algorithms are shown in Table 5.4. In the second column, the number of attribute subsets in the search space is shown. This value

corresponds to the cardinality of the power set of conditional attributes of the decision system. We have sorted the decision systems in ascending order according this value. Then, the number of evaluated candidates for the three algorithms is shown. In Table 5.4, we can corroborate that fast–BR evaluates fewer candidates than fast–CT–EXT and GCreduct in most cases, as we said above. However, since the cost of each evaluation is higher in fast-BR than in GCreduct, the evaluation of fewer candidates does not mean shorter runtime.

Table 5.4: Runtime and candidates evaluated by Fast–CT–EXT, GCreduct and Fast–BR for decision systems taken from the UCI repository.

Decision	Search	Fast–CT–EXT		GCreduct		Fast-BR	
System	space	cands	runtime	cands	runtime	cands	runtime
Credit-g	1.0E6	2.2E5	**0.05**	1.1E5	0.06	1.2E5	0.12
Waveform	2.1E6	1.2E6	2.11	8.1E5	1.88	9.8E5	**1.64**
Flags	5.4E8	2.0E7	1.00	1.2E7	**0.74**	7.1E6	1.06
Student-mat	2.1E9	7.4E8	1003.87	6.4E8	929.46	1.2E8	**81.82**
Student-por	2.1E9	1.2E9	1874.57	1.0E9	1657.90	2.1E8	**161.35**
Dermatology	1.7E10	2.0E8	16.02	1.5E8	12.25	2.3E7	**4.62**
Chess (kr-vs-kp)	6.9E10	6.4E8	7.45	2.9E1	**<0.01**	3.7E3	0.02
Keyword-activity	6.9E10	1.0E8	1.22	2.7E7	**0.42**	1.4E7	0.90
Landsat (train)	6.9E10	4.0E9	23797.99	1.1E10	**9949.31**	1.9E10	17732.49
Cylinder-bands	5.5E11	2.7E7	5.03	2.2E7	4.59	1.2E6	**0.53**
QSAR-biodeg	2.2E12	4.9E7	0.75	7.5E6	**0.19**	3.7E6	0.33
Connect-4	4.4E12	4.0E11	12876.67	1.3E9	**44.23**	9.9E8	160.61
Sponge	3.5E13	2.0E7	0.63	1.5E7	0.58	3.6E5	**0.14**
Diabetes	5.6E14	7.7E8	86.99	1.4E8	**19.48**	8.2E7	23.37
Lung-cancer	7.2E16	3.0E9	188.20	2.2E9	133.43	8.4E7	**7.34**

5.2.3 Finding a Relation Between Some Properties of the Basic Matrix and the Fastest Algorithms for Computing all Reducts

Previous studies [34, 24, 31] categorize the basic matrices depending on the *density of 1's* they have; i.e. the number of ones divided by the number of cells of the basic matrix. Thus, we have selected the density of 1's as the first characteristic of the basic matrix to be studied. In order to explore the performance of GCreduct, fast–CT–EXT and

fast–BR regarding the density of 1's of the basic matrix; we conduct another experiment over 500 randomly generated basic matrices with 2000 rows and 30 columns. The size of these matrices was selected in order to keep reasonable runtime for the three algorithms. The 500 basic matrices were generated with densities of 1's uniformly distributed in the range (0.20–0.80) using a step of 0.04.

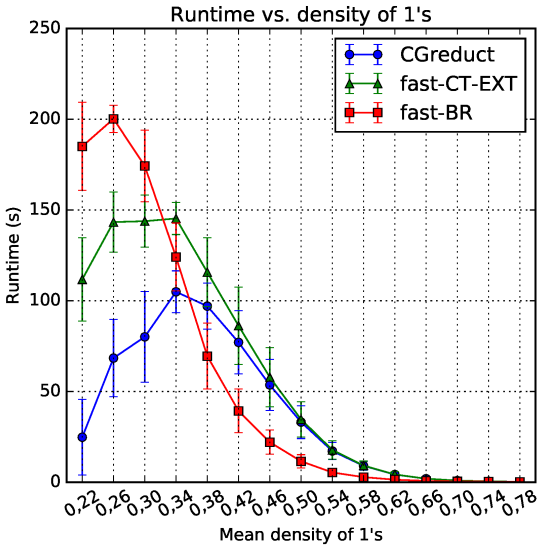

Figure 5.1: Average runtime vs. density of 1's for GCreduct, fast–CT–EXT and fast–BR over all synthetic basic matrices.

For clarity purposes, the 500 synthetic basic matrices were divided into 15 bins by discretizing the range of densities, each bin having approximately 33 basic matrices. Figure 5.1 shows the average runtime for all matrices in each bin for the three algorithms, as a function of the density of 1's in the synthetic basic matrices. In this figure, the vertical bars show the standard deviation into each bin.

From Figure 5.1, we can see that for matrices with density under 0.36 the fastest algorithm was GCreduct, fast–BR was the fastest for matrices with density between 0.36 and 0.66, and the three algorithms show a similar performance for matrices with

density above 0.66. The fact that basic matrices with high density do not constitute a complex computational task for reduct computation [34], is clearly visible in Figure 5.1. For this reason, a detailed analysis in this region lacks of relevance.

In order to explain the line delimiting those basic matrices for which GCreduct is faster than fast–BR (densities under 0.36), we must go deeply into the main difference between these algorithms. The evaluation of the attribute exclusion is the part with the highest complexity within both algorithms ($\Theta(nm)$). This operation is executed more often in fast–BR than in GCreduct. The attribute exclusion occurs when there is at least one column in the sub-matrix of the basic matrix, considering only the attributes in the current candidate that can be removed without increasing the number of zero rows in this sub-matrix. The exclusion is more frequent in matrices with higher density, where overlapping of 1's is more likely. The higher cost of candidate evaluation in fast–BR pays off for basic matrices with higher densities (the limit identified from our experiment is 0.36), because supersets of candidates with attribute exclusion are excluded from subsequent evaluations. For instance, taking the extreme case of the identity matrix, where there is no exclusion at all since every attribute is indispensable to form a reduct. For this kind of basic matrices, GCreduct needs to evaluate as many candidates as fast–BR but the former makes a single verification for exclusion with the set of all attributes. On the other hand, fast–BR verifies the exclusion for each candidate, which leads to higher runtime.

The Friedman and post hoc Nemenyi-Damico-Wolfe-Dunn tests show that GCreduct was significantly faster ($p\text{–}value < 10^{-16}$) than fast–CT–EXT for low (under 0.36) and medium (between 0.36 and 0.66) density matrices. In relation to fast–BR, GCreduct showed a significant runtime reduction ($p\text{–}value < 10^{-16}$) for low density matrices while it was significantly slower ($p\text{–}value < 10^{-16}$) for medium density matrices. From this analysis, we conclude that GCreduct is the best algorithm for low density matrices;

while fast–BR is the best for medium density matrices. For high density matrices (over 0.66), any algorithm can be used since, as we already stated, computing reducts on this kind of matrices does not constitute a complex computational task. Moreover, from these findings a great advantage can be taken of selecting the appropriate algorithm for a specific decision system, since the density of 1's can be computed a priori with a relatively low computational cost.

Table 5.5: Fast–CT–EXT, GCreduct and Fast–BR runtime for decision systems taken from the UCI repository. Sorted by basic matrix density.

Decision system	Density	Fast–CT–EXT runtime (s)	Gcreduct runtime (s)	Fast–BR runtime (s)
Chess (kr-vs-kp)	0.03	7.45	**<0.01**	0.02
Keyword-activity	0.04	1.22	**0.42**	0.90
Connect-4	0.05	12876.67	**44.23**	160.61
QSAR-biodeg	0.12	0.75	**0.19**	0.33
Landsat (train)	0.33	23797.99	**9949.31**	17732.49
Dermatology	0.34	16.02	12.25	**4.62**
Credit-g	0.35	**0.05**	0.06	0.12
Flags	0.35	1.00	**0.74**	1.06
Diabetes	0.38	86.99	**19.48**	23.37
Student-por	0.41	1874.57	1657.90	**161.35**
Sponge	0.42	0.63	0.58	**0.14**
Student-mat	0.43	1003.87	929.46	**81.82**
Lung-cancer	0.47	188.20	133.43	**7.34**
Waveform	0.50	2.11	1.88	**1.64**
Cylinder-bands	0.55	5.03	4.59	**0.53**

In Table 5.5, we included the density of 1's and the runtime shown for each basic matrix in Table 5.4. We also sorted the decision systems of Table 5.5 in ascending order of the density of 1's in their associated basic matrix. Although this is a small heterogeneous sample, the rule obtained from synthetic data can be verified in this table. For decision systems with basic matrix densities close to the boundary of 0.36 we cannot conclude which is the fastest algorithm, as it can be seen in Table 5.5. However, for those decision systems which have a basic matrix with density clearly under 0.36, GCreduct was the fastest. In the same way, fast-BR performed faster for those decision systems having a basic matrix with density clearly above 0.36.

In addition to the density of 1's, other characteristics of the basic matrix were explored such as: the number of reducts, and the standard deviation of 1's by rows and by columns. We found a positive correlation between the number of reducts in the basic matrix and the runtime of the three algorithms. However, this correlation lacks of practical application because the number of reducts cannot be computed a priori. Unfortunately, we did not found a significant correlation between the standard deviation of 1's (both by columns and by rows) and the algorithms' runtime.

5.3 Concluding Remarks

In this chapter, we introduced a new algorithm, GCreduct, for computing all reducts of a decision system. In the proposed algorithm, the search space is traversed evaluating some candidate subsets and, based on pruning properties, many others are discarded. Algorithms reported in the literature use operations with high cost for candidate evaluation, in order to reduce the number of evaluated candidates. The main contribution of our proposal, unlike previous algorithms, is the use of simpler operations for candidate evaluation, based on the pruning properties of gap elimination and attribute contribution, for a faster reduct computation.

After conducting a series of experiments over synthetic basic matrices and decision systems from the UCI repository, we can conclude that GCreduct performs faster than fast–BR and fast–CT–EXT on those decision systems whose associated basic matrix has a density of 1's under 0.36. This result provides a tool to select the algorithm that performs better for a specific decision system.

Chapter 6

A New Algorithm for Computing All the Shortest Reducts

In this chapter, we present a new algorithm, called MinReduct, for computing all the shortest reducts of a decision system. For developing this new algorithm, the search strategy of GCreduct was adapted to compute only the shortest reducts. MinReduct is supported mainly on attribute contribution and binary cumulative operations. In addition, the relation between some properties of the basic matrix and the fastest algorithms for computing the shortest reducts is studied.

The rest of this chapter is structured as follows. In Section 6.1, we introduce the MinReduct algorithm for computing all the shortest reducts of a decision system. In Section 6.2, an evaluation of the proposed algorithm and a discussion of the experimental results are presented. Finally, our concluding remarks appear in Section 6.3.

6.1 MinReduct

In this section, we introduce the MinReduct algorithm for computing all the shortest reducts of a decision system. In addition to the theoretical foundations of CT–EXT and GCreduct, the following proposition, introduced in [59], has a remarkable relevance for computing the shortest reducts.

Proposition 6.1 *Let BM be a basic matrix and B be a subset conditional attributes of a decision system, if B is one of the shortest super–reducts of BM, then it is also one of the shortest reducts of BM.*

In other words, the shortest super–reducts of a decision system are also the shortest

reducts. Notice that every reduct is a super–reduct; and from the condition 2 of the reduct definition, reducts are minimal with respect to inclusion. Proposition 6.1 is specially useful when computing the shortest reducts because a huge amount of exclusion evaluations can be avoided.

In order to reduce the search space, MinReduct arranges the basic matrix in the same way as GCreduct [32].

6.1.1 The MinReduct algorithm

MinReduct finds all the shortest reducts in the basic matrix of a decision system. The key aspects about the proposed algorithm are the use of binary cumulative operations and a fast candidate evaluation process. The search space is pruned such that candidates larger than the shortest reduct found so far are not evaluated. MinReduct searches for super–reducts since the shortest super–reducts are indeed the shortest reducts according to Proposition 6.1.

The MinReduct algorithm traverses the search space in the lexicographical order. A new attribute is added to the current candidate if the candidate is shorter than the shortest super–reducts found so far and the new attribute contributes to the candidate (i.e. the new attribute allows reducing the number of zero rows). If the new attribute contributes, the candidate is evaluated for the super–reduct condition. If it is a super–reduct there are two possibilities: the candidate has the same cardinality of the shortest super–reducts found so far, or the candidate has smaller cardinality than the shortest super–reducts found so far. In the first case the candidate is saved in the result set; in the second case all previously saved super–reducts are removed, and the current candidate is saved as the only element of the result. When the last attribute from the basic matrix is included in the candidate, the algorithm searches for a gap in order to avoid unneeded evaluations. This process continues until the first attribute in the

candidate has 0 in the first row of the basic matrix. Once this condition is reached, the algorithm finishes because there are no remaining reducts (proposition 4.2).

Algorithm 6.1 shows the pseudocode of MinReduct. MinReduct operates over the arranged basic matrix. After initializing the current candidate with the first attribute, the attribute contribution is evaluated by means of Definition 4.2. For those candidates with a contributing attribute, the super–reduct condition is evaluated by means of Definition 4.1. super–reducts, are saved in the result (SR) and if the current candidate has smaller cardinality than the minimum found so far ($maxCard$), the previously stored super–reducts are removed. At this point, the candidate evaluation is finished and the next candidate subset is generated.

The candidate generation process follows the lexicographical order, skipping some unnecessary evaluations when possible. If the current candidate includes the last attribute of the arranged basic matrix, the gap is eliminated. Otherwise, the lexicographical order is followed pruning candidates with a cardinality greater than $maxCard$, and supersets of candidates with a non contributing attribute.

The function *eliminateGap* searches for a gap in the current candidate and eliminates it by skipping unnecessary evaluations as it is proposed in the Corollary 5.1. The function *exclusion* evaluates the exclusion for the current candidate, as described in Propositions 5.2 and 5.3. The function *eliminateOne* generates the next candidate following the lexicographical order. The function $Next$ returns the following attribute in the arranged basic matrix.

An example of the execution of MinReduct over the basic matrix of Table 4.1 is shown in Table 6.1. The first column shows the iteration number corresponding to the evaluated candidate. The second column shows the position of the current candidate in the lexicographical order. It is important to notice that there are 127 attribute subsets in the search space of this example, but only 34 of them are evaluated. In the last

Algorithm 6.1 MinReduct algorithm for computing all the shortest reducts

Input: BM ▷ The arranged basic matrix
c_x ▷ First attribute with a 0 in the first row
c_{max} ▷ Last attribute in the arranged basic matrix
Output: SR ▷ Set of all the shortest reducts

```
1: B ⇐ [ ]                                   ▷ Subset of attributes in the candidate
2: c ⇐ c_0                                   ▷ New attribute to add in the candidate
3: maxCard ⇐ ∞                               ▷ Cardinality of the shortest super–reduct found so far
4: while B ≠ [c_x] do
5:     contributes ⇐ False
6:     superReduct ⇐ False
7:     if Contribution(B + [c]) then
8:         contributes ⇐ True
9:         if SuperReduct(B + [c]) then
10:            superReduct ⇐ True
11:            if |B + [c]| < maxCard then
12:                maxCard ⇐ |B + [c]|
13:                SR ⇐ {B + [c]}             ▷ All previous super–reducts are discarded
14:            else
15:                SR ⇐ SR ∪ {B + [c]}
16:    if c = c_max then                     ▷ Last attribute reached
17:        if superReduct = False then
18:            B + [c] ⇐ eliminateGap(B)
19:        else
20:            if exclusion(B + [c]) = False then
21:                B + [c] ⇐ eliminateGap(B)   ▷ Candidate is a reduct
22:            else
23:                B + [c] ⇐ eliminateOne(B)
24:                maxCard ⇐ maxCard − 1       ▷ A super–reduct that is not a reduct
25:                SR ⇐ ∅
26:    else
27:        if (contributes = True) ∧ (superReduct = False) ∧ (|B + [c]| < maxCard) then
28:            B ⇐ B + [c]
29:        c ⇐ Next(c)
```

column, some comments are included.

Table 6.1: Execution of MinReduct over the basic matrix of Table 4.1.

Iter	Pos	Candidate	Comments
1	1	$[c'_0]$	c'_0 contributes to [ ] but the candidate is not a super–reduct. Add a new attribute.
2	2	$[c'_0, c'_1]$	c'_1 does not contribute to $[c'_0]$. Remove c'_1.
3	34	$[c'_0, c'_2]$	c'_2 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
4	35	$[c'_0, c'_2, c'_3]$	c'_3 contributes to $[c'_0, c'_2]$ but the candidate is not a super–reduct. Add a new attribute.
5	36	$[c'_0, c'_2, c'_3, c'_4]$	The candidate is a super–reduct, it is saved and its cardinality (4) is taken as the new limit for candidates' size.
6	40	$[c'_0, c'_2, c'_3, c'_5]$	The candidate is a super–reduct and it is saved.
7	42	$[c'_0, c'_2, c'_3, c'_6]$	The candidate is a super–reduct. Since it is not a reduct, gap cannot be eliminated. The limit of cardinality is decremented (3). All previous super–reducts are dismissed.
8	43	$[\mathbf{c'_0}, \mathbf{c'_2}, \mathbf{c'_4}]$	The candidate is a super–reduct and it is saved.
9	47	$[c'_0, c'_2, c'_5]$	c'_5 contributes to $[c'_0, c'_2]$ but the limit of candidate's size has been reached. Remove c'_5.
10	49	$[c'_0, c'_2, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_2) is eliminated.
11	50	$[c'_0, c'_3]$	c'_3 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
12	51	$[c'_0, c'_3, c'_4]$	c'_4 contributes to $[c'_0, c'_3]$ but the limit of candidate's size has been reached. Remove c'_4.
13	55	$[c'_0, c'_3, c'_5]$	c'_5 contributes to $[c'_0, c'_3]$ but the limit of candidate's size has been reached. Remove c'_5.
14	57	$[\mathbf{c'_0}, \mathbf{c'_3}, \mathbf{c'_6}]$	The candidate is a super–reduct and it is saved. Since the last attribute is included and the candidate is a reduct, the gap (c'_3) is eliminated.
15	58	$[c'_0, c'_4]$	c'_4 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
16	59	$[c'_0, c'_4, c'_5]$	c'_5 does not contribute to $[c'_0, c'_4]$. Remove c'_5.
17	61	$[\mathbf{c'_0}, \mathbf{c'_4}, \mathbf{c'_6}]$	The candidate is a super–reduct and it is saved. Since the last attribute is included and the candidate is a reduct, the gap (c'_4) is eliminated.

Iter	Pos	Candidate	Comments
18	62	$[c'_0, c'_5]$	c'_5 contributes to $[c'_0]$ but the candidate is not a super–reduct. Add a new attribute.
19	63	$[c'_0, c'_5, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_0) is eliminated.
20	65	$[c'_1]$	c'_1 contributes to [ ] but the candidate is not a super–reduct. Add a new attribute.
21	66	$[c'_1, c'_2]$	c'_2 contributes to $[c'_1]$ but the candidate is not a super–reduct. Add a new attribute.
22	67	$[c'_1, c'_2, c'_3]$	c'_3 contributes to $[c'_1, c'_2]$ but the limit of candidate's size has been reached. Remove c'_3.
23	75	$[c'_1, c'_2, c'_4]$	c'_4 contributes to $[c'_1, c'_2]$ but the limit of candidate's size has been reached. Remove c'_4.
24	79	$[c'_1, c'_2, c'_5]$	c'_5 contributes to $[c'_1, c'_2]$ but the limit of candidate's size has been reached. Remove c'_5.
25	81	$[c'_1, c'_2, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_2) is eliminated.
26	82	$[c'_1, c'_3]$	c'_3 contributes to $[c'_1]$ but the candidate is not a super–reduct. Add a new attribute.
27	83	$[c'_1, c'_3, c'_4]$	c'_4 contributes to $[c'_1, c'_3]$ but the limit of candidate's size has been reached. Remove c'_4.
28	87	$[c'_1, c'_3, c'_5]$	c'_5 contributes to $[c'_1, c'_3]$ but the limit of candidate's size has been reached. Remove c'_5.
29	89	$[c'_1, c'_3, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_3) is eliminated.
30	90	$[c'_1, c'_4]$	c'_4 contributes to $[c'_1]$ but the candidate is not a super–reduct. Add a new attribute.
31	91	$[c'_1, c'_4, c'_5]$	c'_5 contributes to $[c'_1, c'_4]$ but the limit of candidate's size has been reached. Remove c'_5.
32	93	$[c'_1, c'_4, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_4) is eliminated.
33	94	$[c'_1, c'_5]$	c'_5 contributes to $[c'_1]$ but the candidate is not a super–reduct. Add a new attribute.
34	95	$[c'_1, c'_5, c'_6]$	The candidate is not a super–reduct. Since the last attribute is included, the gap (c'_1) is eliminated.
35	97	$[c'_2]$	The algorithm finishes because the column of the basic matrix corresponding to the leftmost attribute in the candidate (c'_2) has a zero in the first row. $SR = \{[c'_0, c'_2, c'_4], [c'_0, c'_3, c'_6], [c'_0, c'_4, c'_6]\}$

In this example, MinReduct starts with the list $B = [\,]$ and $c = c'_0$ such that the first candidate is $B + [c] = [c'_0]$. Clearly the new attribute (c'_0) contributes to $B = [\,]$, therefore the next candidate is $[c'_0, c'_1]$. As it can be seen in Table 4.1, the attribute c'_1 does not contribute to $[c'_0]$ since there are three zero rows in the sub-matrix formed by $[c'_0, c'_1]$ as well as in the sub-matrix formed by $[c'_0]$. Thus, the attribute c'_1 is removed and all the supersets of $[c'_0, c'_1]$ are pruned. Following the lexicographical order, the next candidate is $[c'_0, c'_2]$. Notice that, using Proposition 4.1, we have jumped from the second candidate in the search space to the candidate number 34. c'_2 contributes to $[c'_0]$; then, the next candidate is constructed by keeping c'_2 ($B + [c] = [c'_0, c'_2, c'_3]$). c'_3 contributes to $[c'_0, c'_2]$; thus, the next candidate is $[c'_0, c'_2, c'_3, c'_4]$. This candidate is a super–reduct since there are no zero rows in the sub-matrix formed by the attributes in this list. Thus, $[c'_0, c'_2, c'_3, c'_4]$ is stored in SR and $maxCard$ is updated to four, since there is no reason for analysing greater candidates. The attribute c'_4 is removed to form the next candidate $[c'_0, c'_2, c'_3, c'_5]$ jumping from the 36th to the 40th candidate. This candidate is also a super–reduct and it is saved in SR. The next candidate is $[c'_0, c'_2, c'_3, c'_6]$ because c'_5 has been removed in the same way as c'_4. This new candidate is also a super–reduct and it is also saved. This time, the last attribute has been reached, but the candidate is not a reduct. Since the candidate is a super–reduct but it is not a reduct, the gap cannot be eliminated. Indeed, we have found that there are reducts with a cardinality smaller than $maxCard$; thus, we make $SR = \emptyset$ and $maxCard = maxCard - 1$ (three). The next candidate is $[c'_0, c'_2, c'_4]$, which is the following candidate in the lexicographical order. This candidate is a super–reduct and it is saved. Then, c'_4 is eliminated and $B + [c] = [c'_0, c'_2, c'_5]$. c'_5 contributes to $[c'_0, c'_2]$ but this candidate is not a super–reduct; since the maximum allowed cardinality has been reached, it makes no sense to add a new attribute. Thus, c'_5 is eliminated and $B + [c] = [c'_0, c'_2, c'_6]$. This candidate is not a super–reduct and includes the last attribute. Then, the gap is eliminated to prune

subsets of this candidate. The gap in this case is c'_2 and the next candidate is $[c'_0, c'_3]$. c'_3 contributes to $[c'_0]$; then, we keep c'_3 and $B + [c] = [c'_0, c'_3, c'_4]$. This candidate is not a super–reduct and c'_4 is removed because the maximum cardinality has been reached. Thus, the next candidate is $[c'_0, c'_3, c'_5]$. This candidate is not a super–reduct and again the last attribute is removed; therefore, the next candidate is $[c'_0, c'_3, c'_6]$. The current candidate is a super–reduct (also a reduct) and it is saved in SR. This time, the gap is eliminated to prune subsets of a reduct. The rest of the example in Table 6.1 follows the above described process. The algorithm finishes after the candidate $[c'_1, c'_5, c'_6]$ while evaluating $[c'_2]$, which has a 0 in the first row of the basic matrix.

During the search process, MinReduct evaluates some attribute subsets while discards some others. Propositions 5.1, 4.1 and 4.2 as well as the minimality condition of reducts, guarantee that none of the discarded attribute subsets is a reduct. This is formalized in Proposition 6.2.

Proposition 6.2 *The algorithm MinReduct finds all the shortest reducts of a decision system.*

Proof. *The algorithm MinReduct traverses the search space of attribute subsets in the lexicographical order. During the search process, some attribute subsets are evaluated while other ones are discarded in the following cases:*

- *Supersets of an attribute subset which has a non contributing attribute (Proposition 4.1).*
- *Candidates larger than the shortest super–reduct found so far.*
- *Subsets of a reduct (Proposition 5.1), or a non super–reduct (Corollary 5.1), which includes the last attribute of the basic matrix.*

- *The remaining candidates, when the column of the basic matrix corresponding to the leftmost attribute in the candidate has a zero in the first row (Proposition 4.2).*

Since all attribute subsets discarded by the searching process are certainly not among the shortest reducts, we can state that MinReduct finds all the shortest reducts of a decision system.

6.2 Evaluation and Discussion

In this section, we perform a comparative analysis of the proposed algorithm (MinReduct) versus SRGA [44] and CAMARDF [59]. We have selected SRGA and CAMARDF because they are the fastest algorithms in the state of the art. Since all these algorithms have exponential complexity, we perform a comparison through their implementations. For our experiments, we have implemented MinReduct and SRGA in Java, and we use the implementation of CAMARDF provided by their authors in C.

Evaluations are performed over synthetic basic matrices and real–world datasets taken from the UCI machine learning repository [3]. All experiments were run on a PC with a Core i7-5820K Intel processor at 3.30GHz, with 32GB in RAM, running GNU/Linux.

6.2.1 Finding a Relation Between Some Properties of the Basic Matrix and the Fastest Algorithms for Computing all the Shortest Reducts

For this experiment, we used the same 500 synthetic basic matrices with 2000 rows and 30 columns, that were generated for the experiments of the subsection 5.2.3.

Figure 6.1 shows the average runtime for the three algorithms, as a function of the density of 1's in the synthetic basic matrices. For clarity purposes, the 500 matrices were

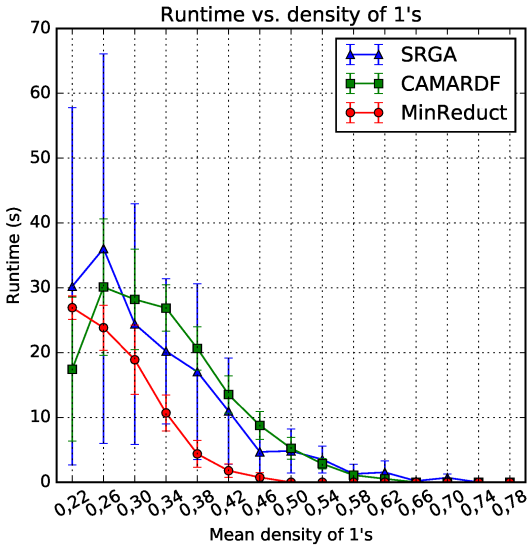

Figure 6.1: Average runtime vs. density of 1's for SRGA, CAMARDF and MinReduct.

divided into 15 bins by discretizing the range of densities, each bin having approximately 33 basic matrices. In this figure, the vertical bars show the standard deviation of each bin.

From Figure 6.1, we notice that CAMARDF was the fastest for matrices with a density under 0.24 while MinReduct was the fastest for matrices with a density above 0.24. Although MinReduct had better performance for most basic matrices, it should be pointed out that for densities of 1's under 0.24, our proposed algorithm showed a not so good performance. This behavior can be explained by looking into the main differences between MinReduct and the other two algorithms. The discernibility function for basic matrices with low density of 1's is very small. Thus, candidate evaluations become very fast in CAMARDF as well as in SRGA, since they iterate over the complete discernibility function. However, for MinReduct, the dimensions of the basic matrix has no relation with its density. Furthermore, for MinReduct there are no simplifications in the candidate evaluation process for low densities of 1's.

In addition to the density of 1's, the correlation between the standard deviation of 1's by rows and by columns, and the algorithms' runtime was studied. Again, we did not found a significant correlation between these properties and the algorithms' runtime.

6.2.2 Evaluation over real datasets

In order to evaluate the relation found before, between the density of a basic matrix and the performance of the algorithms, we present a comparative experiment between our proposed algorithm MinReduct and the other two algorithms over a set of 15 real–world datasets taken from the UCI machine learning repository [3]. For numerical attributes, we used the Weka's equal width discretization method with 10 bins, as describe in [12].

Table 6.2: Datasets taken from the UCI repository sorted according to the density of their basic matrix.

Dataset	Attributes	Instances	Density	Rows	Reducts
Keyword-activity	37	1530	0.04	26	3
Soybean	35	307	0.11	28	359
QSAR-biodeg	42	1055	0.12	40	256
Anneal	38	63	0.21	62	313
Dermatology	35	366	0.34	1103	112708
LED24	25	200	0.34	2458	66800
Student-mat	32	395	0.43	6253	679121
Lung-cancer	57	32	0.47	237	4183355
Arrhythmia	279	452	0.54	52951	-
Optdigits (train)	64	382	0.59	29758	20923529
Landsat (test)	36	2000	0.74	7980	1050755
Ionosphere	34	351	0.74	250	5759
SPECT Heart	22	267	0.90	2284	38473
Ozone	72	575	0.93	5751	82755
Sonar	60	208	0.95	426	3423

In Table 6.2, we show the name of the datasets used in our experiment and their dimensions, as well as the density of 1's and the number of rows of their basic matrix, and their total number of reducts. Datasets in Table 6.2 are sorted in ascending order

Table 6.3: CAMARDF, SRGA and MinReduct runtime for datasets from the UCI repository.

Dataset	Density	Shortest Reducts	Size	CAMARDF runtime (ms)	SRGA runtime (ms)	MinReduct runtime (ms)
Keyword-activity	0.04	1	25	**<1**	3	431
Soybean	0.11	29	11	**<1**	3	284
QSAR-biodeg	0.12	2	13	7	3	278
Anneal	0.21	15	7	**2**	4	16
Dermatology	0.34	137	6	114	**45**	73
LED24	0.34	95	11	1368	**521**	540
Student-mat	0.43	6	18	530	178	**155**
Lung-cancer	0.47	112	4	619	26	**25**
Arrhythmia	0.54	405	2	-	2632	**57**
Optdigits (train)	0.59	600	5	48368	15492	**3754**
Landsat (test)	0.74	432	5	-	1815	**131**
Ionosphere	0.74	10	2	3	3	**1**
SPECT Heart	0.90	3551	3	-	747	**15**
Ozone	0.93	239	2	117	193	**6**
Sonar	0.95	1222	2	-	48	**2**

regarding the density of their basic matrix.

In Table 6.3, we show the number and cardinality of the shortest reducts for each dataset, as well as the runtime of the three algorithms. In this experiment, CAMARDF ran out of memory for four datasets (denoted as "-"). From this experiment, we conclude that the rule obtained for synthetic matrices is also satisfied by real datasets, i.e. CAMARDF and SRGA are faster for basic matrices with low density of 1's, otherwise MinReduct is the fastest.

For Arrhythmia the total number of reducts could not be computed after 1500 hours. However, all the shortest reducts were found in a very small time by MinReduct, as it can be seen in Table 6.3. This runtime reduction makes the shortest reduct computation useful for larger datasets, specially those with high dimensionality, for which computing all reducts is unfeasible.

6.3 Concluding Remarks

In this chapter, we introduced a new algorithm, MinReduct, for computing all the shortest reducts of a decision system. The proposed algorithm uses binary cumulative operations and a fast candidate evaluation process. Previous algorithms, reported in the literature, operate over the discernibility function and relay on high cost operations for generating and evaluating candidates.

After conducting a series of experiments over synthetic basic matrices and real datasets from the UCI machine learning repository, we can conclude that MinReduct performs faster than the fastest previously reported algorithms on those datasets whose associated basic matrix has a density of 1's above 0.24. This result provides also a tool to select the fastest algorithm for a specific dataset.

Chapter 7

Conclusions

The notion of reduct is an important concept within Rough Set Theory. Reducts are minimal subsets of attributes preserving the discernibility capacity of the whole set of attributes in a decision system. Unfortunately, computing all reducts of a decision system is NP–hard, and thus several approximate algorithms have been proposed to overcome the high complexity of this problem. However, approximate algorithms do not return the complete set of reducts, and sometimes they obtain non minimal subsets. Hence, the development of exact algorithms for speeding up reduct computation is an active research topic.

From our literature review, we found that hardware implementations had been reported as the fastest approach to reduct computation. Thus, as a first step in this PhD research, a new hardware platform for computing all reducts of a decision system was proposed. Our proposed platform outperforms previous hardware and software implementations in terms of runtime. However, the size of the problem that can be solved by hardware platforms is significantly limited by the FPGA resources; which reduces their practical application. Therefore, the PhD research was directed to the development of new algorithms for speeding up reduct computation. In this direction, a new algorithm for computing all reducts of a decision system that uses fast operations for candidate evaluation, which allows reducing the runtime for a specific kind of decision systems, was introduced. In addition, an experimental study for finding a relation between some properties of a decision system and the fastest algorithms for computing all reducts, was presented.

Furthermore, a new algorithm for computing all the shortest reducts was introduced.

Our proposed algorithm is competitive with other algorithms in the state–of–the–art for the general case, and faster for a specific kind of decision systems. Also, an experimental study on the relation between some properties of a decision system and the fastest algorithms for computing the shortest reducts, was performed.

The rest of this chapter is structured as follows. Section 7.1 presents the conclusions of this PhD research. In Section 7.2, our contributions are shown. Section 7.3 contains some directions for future work. Finally, in Section 7.4, the publications derived from this PhD research are listed.

7.1 Conclusions

The general objective of this PhD research, regarding the development of new algorithms for computing reducts of decision systems; which should be comparable to state of the art algorithms in most datasets, and faster in some specific kinds of datasets, was successfully accomplished.

Regarding our proposed hardware architecture for computing all reducts on decision systems, based on our experimental results, we can conclude that:

- The proposed platform, based on the CT–EXT algorithm [36], uses fewer hardware resources and it is able to run at higher clock frequency than the hardware implementation of BT [34]. This characteristic allows processing larger matrices. Furthermore, the proposed platform is the fastest hardware architecture for computing all reducts of a decision system. This result meets the specific objective of developing a new algorithm for computing all reducts.

- Hardware platforms for reduct computation are faster for basic matrices big enough to overcome the delay of the synthesis process and to take advantage of the single clock candidate evaluation; but the size of these basic matrices is

limited by the FPGA resources. This issue drastically reduces their practical application.

Regarding our proposed algorithm for computing all reducts of a decision system, based on our experimental results, we can conclude that:

- Our proposed algorithm GCreduct is the fastest algorithm for computing all reducts on those decision systems whose associated basic matrix has a density of 1's lower than 0.36. This result meets the specific objective of developing a new algorithm for computing all reducts.

- We found a relation between the density of 1's in the basic matrix and the fastest algorithms for computing all reducts. This relation can be used as an effective tool for selecting a priori the best algorithm for a specific decision system. This result meets the specific objective of finding a relation between some properties of the basic matrix and the fastest algorithms for computing all reducts.

Regarding our proposed algorithm for computing all the shortest reducts of a decision system, based on our experimental results, we can conclude that:

- Our proposed algorithm MinReduct performs faster than the fastest previously reported algorithms on those decision systems whose associated basic matrix has a density of 1's higher than 0.24. This result meets the specific objective of developing a new algorithm for computing all the shortest reducts.

- We also found a relation between the density of 1's in the basic matrix and the fastest algorithms for computing all the shortest reducts. This relation can be used as an effective tool for selecting a priori the best algorithm for computing all the shortest reducts of a specific decision system. This result meets the specific objective of finding a relation between some properties of the basic matrix and the fastest algorithms for computing all the shortest reducts.

7.2 Contributions

The contributions of this PhD research are the following:

- A new hardware–software platform for computing all reducts based on the CT–EXT algorithm; which performs faster, and is able to process larger problems, than previous hardware platforms for computing all reducts.

- A new algorithm (GCreduct) for computing all reducts based on a fast candidate evaluation process. This algorithm is the fastest alternative for reduct computation on those decision systems whose associated basic matrix has low density of 1's.

- A relation based on the density of 1's of the basic matrix for determining a priori the fastest algorithm for computing all reducts of a specific decision system.

- A new algorithm (MinReduct) for computing all the shortest reducts based on a fast candidate evaluation process. This algorithm is the fastest alternative for reduct computation on those decision systems whose associated basic matrix has medium or high density of 1's.

- A relation based on the density of 1's of the basic matrix for determining a priori the fastest algorithm for computing all the shortest reducts of a specific decision system.

7.3 Future work

The results obtained in this PhD research open further studies on exact algorithms for computing reducts. As future work, we proposed the following:

- The amount of resources required by hardware architectures for reduct computation determines the maximum size of the basic matrix that can be processed. Thus, the search for new algorithms that could be efficiently (requiring fewer resources) implemented on an FPGA, constitutes an important direction for future work.

- New ways for improving hardware architectures, such as testing two or more candidates at the same time, are still unexplored and should be evaluated.

- Further studies would include a deeper exploration of the relation between some other dataset's properties and the performance of different strategies for computing reducts. For example, the rank of the basic matrix, its dimension and the minimum number of 1's in a row, were not considered in this research. We think that the selection of the appropriate algorithm for a given dataset can drastically reduce the runtime.

- Another interesting line for future research is the development of a meta–algorithm that uses the relations found in this research for switching the search strategy through the searching process as a function of the basic–matrix's properties.

7.4 Publications

The following publications were derived from this PhD research.

JCR Journals:

- Rodríguez-Diez, V. et al. (2020). MinReduct: A new algorithm for computing the shortest reducts. *Pattern Recognition Letters*, 138, 177–184. [IF: 3.255, **Q1**]
- Rodríguez-Diez, V. et al. (2018). A new algorithm for reduct computation based on gap elimination and attribute contribution. *Information Sciences*, 435, 111–123. [IF: 4.832, **Q1**]
- Rodríguez-Diez, V. et al. (2015). A fast hardware software platform for computing irreducible testors. *Expert Systems with Applications*, 42(24), 9612–9619. [IF: 3.928, **Q1**]

Conference Proceedings:

- Rodríguez-Diez, V. et al. A Comparative Study of Two Algorithms for Computing the Shortest Reducts: MiLIT and MinReduct. *Lecture Notes in Computer Science* 12725, 57–67, 2021.
- Rodríguez-Diez, V. et al. The Impact of Basic Matrix Dimension on the Performance of Algorithms for Computing Typical Testors. *Lecture Notes in Computer Science* 10880, 41–50, 2018.
- Rodríguez-Diez, V. et al. Fast–BR vs. Fast–CT_EXT: An Empirical Performance Study. *Lecture Notes in Computer Science* 10267, 127–136, 2017.
- Rodríguez-Diez, V. et al. A hardware architecture for filtering irreducible testors. *In ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, 1–4, 2014.

Book Chapter:

- Rodríguez-Diez, V. et al. (2017). Algorithms for Computing Rough Set Reducts. In *Combinatorial Algorithms and Learning* (13–23). Montiel & Soriano Editores, BUAP.

Technical Report:

- Rodríguez-Diez, V. & Martínez-Trinidad, J. F. Development of fast algorithms for reduct computation (Report No. CCC–16–005). *Puebla, Mexico: Instituto Nacional de Astrofísica, Óptica y Electrónica*, 1–44, 2016.

Unpublished:

- Rodríguez-Diez, V. et al. A Review on Algorithms for Computing Reducts. *To be summited to Artificial Intelligence Review.* [IF: 2.627, **Q1**]

Bibliography

[1] Alba-Cabrera, E., Ibarra-Fiallo, J., Godoy-Calderon, S., and Cervantes-Alonso, F. (2014). YYC: A Fast Performance Incremental Algorithm for Finding Typical Testors. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 416–423. Springer.

[2] Alba-Cabrera, E., Santana, R., Ochoa, A., and Lazo-Cortés, M. (2000). Finding typical testors by using an evolutionary strategy. In *Proceedings of the Fith Ibero American Symposium on Pattern Recognition*, volume 0, pages 267–278.

[3] Bache, K. and Lichman, M. (2013). UCI machine learning repository.

[4] Chen, Y., Miao, D., and Wang, R. (2010). A rough set approach to feature selection based on ant colony optimization. *Pattern Recognition Letters*, 31(3):226–233.

[5] Chen, Y., Zhu, Q., and Xu, H. (2015). Finding rough set reducts with fish swarm algorithm. *Knowledge-Based Systems*, 81:22–29.

[6] Chikalov, I., Lozin, V. V., Lozina, I., Moshkov, M., Nguyen, H. S., Slowron, A., and Zielosko, B. (2013). *Three approaches to data analysis*. Springer Science & Business Media.

[7] Chouchoulas, A. and Shen, Q. (2001). Rough set-aided keyword reduction for text categorization. *Applied Artificial Intelligence*, 15(January):843–873.

[8] Cumplido, R., Carrasco-Ochoa, J. A., and Feregrino, C. (2006). On the Design and Implementation of a High Performance Configurable Architecture for Testor Identification LNCS.pdf. In *CIARP 2006*, pages 665–673.

[9] Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397.

[10] Digilent Inc. (2010). Digilent Synchronous Parallel Interface (DSTM) Programmer's Reference Manual.

[11] Digilent Inc. (2013). Atlys Board Reference Manual.

[12] Flores, M., Gámez, J., Martínez, A., and Puerta, J. (2010). Analyzing the impact of the discretization method when comparing Bayesian classifiers. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6096 LNAI, pages 570–579.

[13] Grze, T. (2013). FPGA in Rough Set Based Core and Reduct Computation. pages 263–270.

[14] Jensen, R. and Shen, Q. (2003). Finding rough set reducts with ant colony optimization. *Proceedings of the 2003 UK workshop on*, 1(2):15–22.

[15] Jensen, R. and Shen, Q. (2004). Semantics-preserving dimensionality reduction: Rough and fuzzy-rough-based approaches. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1457–1471.

[16] Jensen, R., Tuson, A., and Shen, Q. (2014). Finding rough and fuzzy-rough set reducts with SAT. *Information Sciences*, 255:100–120.

[17] Jiang, Y. and Yu, Y. (2015). Minimal attribute reduction with rough set based on compactness discernibility information tree. *Soft Computing*, (2009):1–11.

[18] Jiao, N., Miao, D., and Zhou, J. (2010). Two novel feature selection methods based on decomposition and composition. *Expert Systems with Applications*, 37(12):7419–7426.

[19] Kopczynski, M., Grze, T., and Stepaniuk, J. (2014). FPGA in Rough-Granular Computing: Reduct Generation. In *International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, pages 364–370.

[20] Lazo-Cortés, M., Ruiz-Shulcloper, J., and Alba-Cabrera, E. (2001). An overview of the evolution of the concept of testor. *Pattern Recognition*, 34(4):753–762.

[21] Lazo-Cortés, M. S., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., and Sánchez-Díaz, G. (2013). Easy categorization of attributes in decision tables based on basic binary discernibility matrix. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8258 LNCS(PART 1):302–310.

[22] Lazo-Cortés, M. S., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., and Sanchez-Diaz, G. (2015). On the relation between rough set reducts and typical testors. *Information Sciences*, 294:152–163.

[23] Lias-Rodríguez, A. and Pons-Porrata, A. (2009). BR: A new method for computing all typical testors. In *CIARP 2009*, volume 5856, pages 433–440.

[24] Lias-Rodríguez, A. and Sanchez-Diaz, G. (2013). An Algorithm for Computing Typical Testors Based on Elimination of Gaps and Reduction of Columns. *International Journal of Pattern Recognition and Artificial Intelligence*, 27(08):1350022.

[25] Lin, T. Y. and Yin, P. (2004). Heuristically Fast Finding of the Shortest Reducts. In *Rough Sets and Current Trends in Computing*, pages 465–470. Springer Berlin Heidelberg.

[26] Parthaláin, N. M., Jensen, R., and Shen, Q. (2008). Finding fuzzy-rough reducts with fuzzy entropy. *IEEE International Conference on Fuzzy Systems*, pages 1282–1288.

[27] Pawlak, Z. (1981). *Classification of objects by means of attributes*. Polish Academy of Sciences [PAS]. Institute of Computer Science.

[28] Piza-Davila, I., Sanchez-Diaz, G., Aguirre-Salado, C. A., and Lazo-Cortes, M. S. (2014). A parallel hill-climbing algorithm to generate a subset of irreducible testors. *Applied Intelligence*, 42(4):622–641.

[29] Piza-Davila, I., Sanchez-Diaz, G., Lazo-Cortes, M. S., and Rizo-Dominguez, L. (2017). A CUDA-based hill-climbing algorithm to find irreducible testors from a training matrix. *Pattern Recognition Letters*, 95:22–28.

[30] Rodríguez, V., Martínez, J. F., Carrasco, J. A., Lazo, M. S., Cumplido, R., and Feregrino Uribe, C. (2014). A hardware architecture for filtering irreducible testors. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–4. IEEE.

[31] Rodríguez-Diez, V., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., Lazo-Cortés, M., Feregrino-Uribe, C., and Cumplido, R. (2015). A fast hardware software platform for computing irreducible testors. *Expert Systems with Applications*, 42(24):9612–9619.

[32] Rodríguez-Diez, V., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., and Lazo-Cortés, M. S. (2018). A new algorithm for reduct computation based on gap elimination and attribute contribution. *Information Sciences*, 435:111–123.

[33] Rojas, A. and Cumplido, R. (2007). FPGA-based architecture for computing testors. In *IDEAL 2007*, pages 188–197.

[34] Rojas, A., Cumplido, R., Ariel Carrasco-Ochoa, J., Feregrino, C., and Francisco Martínez-Trinidad, J. (2012). Hardware-software platform for computing irreducible testors. *Expert Systems with Applications*, 39(2):2203–2210.

[35] Ruiz-Shulcloper, J., Aguila, L., and Bravo, A. (1985). BT and TB algorithms for computing all irreducible testors. *Mathematics Sciences Journal (In Spanish)*, 2:11–18.

[36] Sanchez, G. and Lazo, M. (2007). CT-EXT: an algorithm for computing typical testor set. In *Progress in Pattern Recognition, Image Analysis and Applications*, pages 506–514. Springer.

[37] Sanchez-Diaz, G., Diaz-Sanchez, G., Mora-Gonzalez, M., Piza-Davila, I., Aguirre-Salado, C. a., Huerta-Cuellar, G., Reyes-Cardenas, O., and Cardenas-Tristan, A. (2014). An evolutionary algorithm with acceleration operator to generate a subset of typical testors. *Pattern Recognition Letters*, 41(1):34–42.

[38] Sánchez-Diaz, G., Piza-Davila, I., Lazo-Cortés, M., Mora-González, M., and Salinas-Luna, J. (2010). A fast implementation of the CT-EXT algorithm for the testor property identification. In *LNAI 2010*, volume 6438, pages 92–103.

[39] Santiesteban, Y. and Pons-Porrata, A. (2003). LEX: a new algorithm for the calculus of typical testors. *Mathematics Sciences Journal (In Spanish)*, 21(1):85–95.

[40] Skowron, A. and Rauszer, C. (1992). The discernibility matrices and functions in information systems. In *Intelligent Decision Support*, pages 331–362. Springer.

[41] Starzyk, J., Nelson, D. E., and Sturtz, K. (1999). Reduct generation in information systems. *Bulletin of international rough set society*, 3(May):19–22.

[42] Starzyk, J. a., Nelson, D. E., and Sturtz, K. (2000). A Mathematical Foundation for Improved Reduct Generation in Information Systems. *Knowledge and Information Systems*, 2(2):131–146.

[43] Strakowski, T. and Rybiński, H. (2008). A New Approach to Distributed Algorithms for Reduct Calculation. *Transactions on Rough Sets IX*, (3):365–378.

[44] Susmaga, R. (1998). COMPUTATION OF SHORTEST REDUCTS.

[45] Susmaga, R. (1999). Computation of minimal cost reducts. In *Raś Z.W., Skowron A. (eds) Foundations of Intelligent Systems. ISMIS 1999*, volume 1609, pages 448–456. Springer, Berlin, Heidelberg.

[46] Tiwari, K. and Kothari, A. (2011). Architecture and Implementation of Attribute Reduction Algorithm Using Binary Discernibility Matrix. In *2011 International Conference on Computational Intelligence and Communication Networks*, pages 212–216.

[47] Tiwari, K., Kothari, A., and Keskar, A. (2012). Reduct generation from binary discernibility matrix: an hardware approach. *International Journal of Future Computer and Communication*, 1(3):270–272.

[48] Tiwari, K., Kothari, A., and Shah, R. (2013). FPGA Implementation of a Reduct Generation Algorithm based on Rough Set Theory. *International Journal of Advanced Electrical and Electronics Engineering (IJAEEE)*, 2(6).

[49] Tiwari, K. S. (2014). Design and Implementation of Rough Set Algorithms on FPGA : A Survey. 3(9):14–23.

[50] Wang, J. and Wang, J. (2001). Reduction algorithms based on discernibility matrix: the ordered attributes method. *Journal of computer science and technology*, 16(6):489–504.

[51] Wang, P.-C. (2007). Highly Scalable Rough Set Reducts Generation. *Journal of Information Science and Engineering*, 4(23):1281–1298.

[52] Wang, X., Yang, J., Teng, X., Xia, W., and Jensen, R. (2007). Feature selection based on rough sets and particle swarm optimization. *Pattern Recognition Letters*, 28(4):459–471.

[53] Wroblewski, J. (1995). Finding minimal reducts using genetic algorithms. In *Proccedings of the second annual join conference on infromation science*, pages 186–189.

[54] Xilinx Inc. (2012). LogiCORE IP FIFO Generator v9.2.

[55] Yang, P. Y. P., Li, J. L. J., and Huang, Y. H. Y. (2008). An Attribute Reduction Algorithm by Rough Set Based on Binary Discernibility Matrix. *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, 2.

[56] Yao, Y. and Zhao, Y. (2009). Discernibility matrix simplification for constructing attribute reducts. *Information Sciences*, 179(7):867–882.

[57] Zheng, K., Hu, J., Zhan, Z., Ma, J., and Qi, J. (2014). An enhancement for heuristic attribute reduction algorithm in rough set. *Expert Systems with Applications*, 41(15):6748–6754.

[58] Zhong, N., Dong, J., and Ohsuga, S. (2001). Using rough sets with heuristics for feature selection. *Journal of intelligent information systems*, 16(3):199–214.

[59] Zhou, J., Miao, D., Feng, Q., and Sun, L. (2009). Research on Complete Algorithms for Minimal Attribute Reduction. In *RSKT 2009*, pages 152–159.