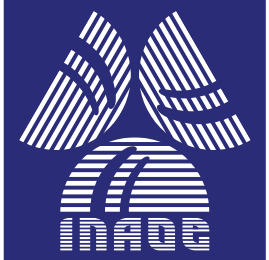


Algoritmo para la búsqueda de todos los reductos más cortos

por

Yanir González Díaz

Disertación presentada en cumplimiento parcial
de los requisitos para
el grado de

MSc. en Ciencias Computacionales

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica ***(INAOE)***
Tonantzintla, Puebla, Mexico
abril, 2021

Tutores:

Dr. José Francisco Martínez Trinidad
Dr. Jesús Ariel Carrasco Ochoa
Coordinación de Ciencias Computacionales
INAOE, Mexico

Dr. Manuel S. Lazo Cortés
TecNM/Instituto Tecnológico de Tlalnepantla
ITTLA, Mexico

Abstract

In Rough Set Theory (RST) a supervised classification problem is represented through Decision Systems. A Decision System is a table in which the rows and columns represent objects and attributes respectively of a data set. On the other hand, the minimum subsets of attributes that preserve the ability to discern the set of objects in the decision system are called reducts. The problem of computing all reducts has an exponential complexity with respect to the number of attributes in a decision system. Therefore, some authors have chosen to calculate only the shortest reducts. In the literature, some algorithms for the computation of all shortest reducts have been reported. However, most of these algorithms are based on time consuming operations for the evaluation of reducts candidates. In this thesis, an algorithm to calculate all the shortest reducts in a decision system, which is based on estimating the size of the shortest reducts and some pruning strategies based on the concepts of core and contribution of an attribute, is introduced. In the proposed algorithm, first, the size of the shortest reducts is approximated, then starting from this approximation and performing some pruning of the search space, the real size of the shortest reducts is determined. Finally, knowing the real size of the shortest reducts and also pruning the search space all the shortest reducts are computed. The results of the experiments over several synthetic and real-world decision systems show that the proposed algorithm is faster than the algorithms reported in the literature in most simplified binary discernibility matrices.

Keywords: **Shortest Reducts, Rough Sets, Decision Systems**.

Resumen

En la Teoría de Conjuntos Rugosos *(TCR)* un problema de clasificación supervisado se representa a través de los Sistemas de Decisión. Un Sistema de Decisión es una tabla en la cual las filas y las columnas representan objetos y atributos, respectivamente, de un conjunto de datos. Por otro lado, los subconjuntos mínimos de atributos que conservan la capacidad de discernir al conjunto de objetos en el sistema de decisión son denominados reductos. El cálculo de todos los reductos en un sistema de decisión tiene una complejidad exponencial con respecto al número de atributos. Por lo tanto, algunos autores han optado por calcular solo los reductos más cortos. En la literatura existen algunos algoritmos para calcular todos los reductos más cortos. Sin embargo, la mayoría de estos algoritmos se basan en operaciones costosas para la evaluación de candidatos a reductos. En el algoritmo propuesto, primero, se aproxima el tamaño de los reductos más cortos, luego, a partir de esta aproximación y realizando una poda del espacio de búsqueda, se determina el tamaño real de los reductos más cortos. Finalmente, conociendo el tamaño real de los reductos más cortos y también podando el espacio de búsqueda, se calculan todos los reductos más cortos. Los resultados de los experimentos realizados sobre sistemas de decisión sintéticos y reales, muestran que el algoritmo propuesto es más rápido que los algoritmos reportados en la literatura, en la mayoría de las matrices de discernibilidad binaria simplificada.

Palabras Clave: Reductos más cortos, Conjuntos Rugosos, Sistemas de Decisión.

Agradecimientos

A mi familia, por la confianza y cariño, siempre estando conmigo en todo momento, a quienes les dedico esta satisfacción de concluir este ciclo importante en mi vida.

A mis asesores, Dr. Manuel S. Lazo Cortés, Dr. José Francisco Martínez Trinidad y Dr. Jesús Ariel Carrasco Ochoa quienes con su apoyo, paciencia, pero sobre todo su conocimiento y experiencia han guiado y han sido parte medular de este proyecto, y mi progreso académico.

A todas aquellas personas que de alguna forma u otra ayudaron a la realización de esta tesis.

Al Consejo Nacional de Ciencia y Tecnología *(CONACYT)* y al INAOE por todas las facilidades prestadas durante mi estancia académica.

Muchas Gracias,
Yanir.
Tonantzintla, Puebla, Mexico.
abril, 2021.

Índice general

Índice de figuras ix

Índice de tablas xi

Lista de Algoritmos xiii

Acrónimos xv

1. Introducción 1

1.1. Motivación . . . 1
1.2. Descripción del Problema . . . 2
1.3. Objetivos . . . 3
1.4. Organización de la tesis . . . 4

2. Conceptos Básicos 5

2.1. Sistema de Decisión . . . 5
2.2. Relación de Indiscernibilidad . . . 6
2.3. Reducto y Core . . . 7
2.4. Matriz de Discernibilidad . . . 8
2.5. Matriz de Discernibilidad Binaria . . . 9
2.6. Matriz de Discernibilidad Binaria Simplificada . . . 11
2.7. Contribución de un atributo . . . 12
2.8. Síntesis . . . 13

3. Trabajo Relacionado 15

3.1. Cálculo de reductos más cortos . . . 16
3.2. Discusión . . . 19
3.3. Síntesis . . . 21

4. Algoritmo para el cálculo de los reductos más cortos **23**
4.1. Estimación del tamaño de los reductos más cortos 25
4.2. Determinación del tamaño de los reductos más cortos 28
4.2.1. Cálculo del core 28
4.2.2. Partición del conjunto de atributos 30
4.2.3. Poda del espacio de búsqueda 39
4.2.4. Determinación de super-reductos de tamaño K 46
4.2.5. Algoritmo para determinar el tamaño de los reductos más cortos 48
4.3. Cálculo de todos los reductos más cortos 50
4.4. Análisis de Complejidad 53
4.5. Síntesis 56

5. Experimentos **57**
5.1. Descripción de los conjuntos de datos 57
5.2. Evaluación del algoritmo propuesto 58
5.2.1. Estimación del tamaño de los reductos más cortos 59
5.2.2. Selección del número de candidatos a generar 62
5.2.3. Comparación con el estado del arte 63
5.3. Síntesis 69

6. Conclusiones **71**
6.1. Trabajo Futuro 73

Bibliografía **74**

Índice de figuras

3.1. Taxonomía de los algoritmos para el cálculo de reductos 15

4.1. Estrategia del algoritmo propuesto. 24
4.2. Corrección de la estimación del tamaño de los reductos más cortos. . . 38
4.3. Estrategia utilizada para determinar el tamaño real de los reductos más cortos. 38

5.1. Estimación del tamaño de los reductos más cortos realizada por el mejor modelo de regresión lineal obtenido. 61
5.2. Comparación del tiempo de ejecución en segundos y los requerimientos máximos de memoria de las estrategias propuestas en [Piza-Dávila et al., 2020] y el algoritmo propuesto. 65
5.3. A la izquierda se muestra el tiempo de ejecución en segundos de los algoritmos y a la derecha se muestra los requerimientos máximos de memoria en Megabytes de los mismos, en donde *MiLIT* es el algoritmo introducido en [Piza-Dávila et al., 2020] y MiLIT* es la mejor estrategia (*PFRC-MiLIT* y *NC-MiLIT*) para cada MDBS. 66

Índice de tablas

2.1. Sistema de Decisión. 6
2.2. Matriz de Discernibilidad del SD de Tabla 2.1. 9
2.3. Matriz de Discernibilidad Binaria del SD de la Tabla 2.1. 10
2.4. Matriz de Discernibilidad Binaria Simplificada del SD de la Tabla 2.1. . 12

4.1. Valores de las características extraídas para algunas $MDBS$. 26
4.2. Matriz de Discernibilidad Binaria Simplificada. 30
4.3. Ejemplo de ejecución del Algoritmo 4.2, usando la MDBS de la Tabla 4.2. 33
4.4. Matriz de Discernibilidad Binaria Simplificada ordenada asociada a la MDBS de la Tabla 4.2 42

5.1. Descripción de los conjuntos de datos utilizados. 58
5.2. Estimación del tamaño de los reductos más cortos realizada por el mejor modelo de regresión lineal obtenido. 60
5.3. Tiempos de ejecución en segundos del algoritmo propuesto para algunos conjuntos de datos del repositorio UCI, considerando diferentes tamaños para el número de candidatos a generar. 63
5.4. Tiempo de ejecución de los algoritmos en horas, minutos y segundos, en donde PFRC-MiLIT y NC-MiLIT son las estrategias introducidas en [Piza-Dávila et al., 2020], resaltando en color rojo el tiempo de la estrategia que escoge el algoritmo *MiLIT*. 67
5.5. Conjuntos de datos en los cuales las diferencias entre el tiempo de ejecución del algoritmo propuesto y el algoritmo MiLIT son más significativas. 68

Lista de Algoritmos

4.1. Generar core 29
4.2. Partición del conjunto de atributos 33
4.3. Generación de candidatos a super-reductos 45
4.4. Cómputo del tamaño de los reductos más cortos 49
4.5. Búsqueda de todos los reductos más cortos 52

Acrónimos

MDB	Matriz de Discernibilidad Binaria
MDBS	Matriz de Discernibilidad Binaria Simplificada
TCR	Teoría de Conjuntos Rugosos
CMA	Conjunto mínimo de atributos
CR	Conjuntos Rugosos
SD	Sistema de Decisión

Capítulo 1

Introducción

1.1 Motivación

Una herramienta matemática utilizada para el análisis de datos es la Teoría de Conjuntos Rugosos *(TCR)*, propuesta por Z. Pawlak y su equipo [Pawlak, 1982]. El propósito principal de la TCR es la transformación de los datos en conocimiento, es decir, se trata de extraer información útil de los datos. Esta teoría permite establecer una metodología para la extracción de patrones dentro de un conjunto de datos y está basada en los conceptos de "discernibilidad" y "aproximación". Discernir significa "distinguir una cosa de otra", lo que se busca es encontrar todos aquellos objetos que son discernibles. Aproximación se refiere a la existencia de vaguedad e imprecisión en la información de un conjunto de objetos.

La filosofía de los Conjuntos Rugosos *(CR)* se basa en asumir que existe información asociada con cada objeto del universo de discurso. La información se representa mediante una tabla donde cada fila representa un objeto y cada columna un atributo, a lo cual se le denomina sistema de información. Un sistema de decisión es un sistema de información donde un atributo indica a qué clase pertenece cada objeto. Uno de los conceptos principales en la TCR es la noción de *reducto*, el cual se puede definir como un subconjunto mínimo de atributos que permiten discernir entre objetos que pertenecen a diferentes clases, con igual capacidad que el conjunto completo de atributos. En la TCR, los reductos se usan comúnmente en problemas de reducción de dimensionalidad [Carreira-Perpinán, 1997], selección de características [An et al., 2004; Siedlecki and Sklansky, 1993] y para construir clasificadores basados en reglas [Lazo-Cortés et al.,

2018]. Algunas de las aplicaciones prácticas de la TCR son: Ensamble de clasificadores basados en la TCR para clasificación de páginas web [Saha et al., 2007], uso de reductos para el análisis de retroalimentación de relevancia del usuario en la recuperación de información de texto [Singh and Prasad, 2008], predicción de enfermedad cardíaca utilizando *random forest* y selección de características basadas en conjuntos aproximados [Yekkala and Dixit, 2018], entre otras. Sin embargo, la principal restricción en las aplicaciones prácticas de la TCR es que calcular todos los reductos de un sistema de información es un problema NP-dificil [Rauszer and Skowron, 1992] y seleccionar el 'mejor reducto' entre el conjunto completo de todos los reductos es costoso, ya que puede haber muchos reductos en un sistema de decisión. Por lo tanto, algunos autores han optado por calcular solo los reductos más cortos [Susmaga, 1998; Lin and Yin, 2004], una línea de investigación activa es el desarrollo de algoritmos rápidos para el cálculo de todos los reductos más cortos.

1.2 Descripción del Problema

La búsqueda exhaustiva en el cálculo de reductos no es una solución factible, incluso en problemas con un número relativamente pequeño de atributos, ya que si contamos con un sistema de decisión con N atributos se tienen $2^N - 1$ posibles subconjuntos de atributos a evaluar. Existen dos estrategias básicas para realizar el cálculo de los reductos: la estrategia incremental y la decremental [Yao et al., 2008], en el primer caso se parte del conjunto vacío y se van adicionando atributos en cada paso hasta que se forme un reducto, por otra parte, en la segunda variante, se parte del conjunto de todos los atributos y se van eliminando atributos hasta que se llegue a un reducto. Para realizar el cálculo de los reductos sin realizar una búsqueda exhaustiva se han

utilizado algunas heurísticas basadas en: importancia de los atributos [Hu, 1995], información mutua [Wang et al., 2004] y algoritmos genéticos [Wroblewski, 1995; Bazan et al., 2000], entre otras. El principal inconveniente de este enfoque es que estos algoritmos no necesariamente devuelven el conjunto completo de reductos de un sistema de información, y algunas veces pueden obtener super-reductos, es decir, subconjuntos que tienen la misma capacidad de discernibilidad que todo el conjunto de atributos condicionales pero no necesariamente son irreducibles. No obstante para algunas aplicaciones no son necesarios todos los reductos [Yu et al., 2009; Grzymala-Busse, 2005], en algunos estudios se ha evaluado la factibilidad de calcular todos los reductos más cortos (reductos de menor cardinalidad), en lugar de calcularlos todos [Sil and Das, 2012]. En la literatura existen muy pocos algoritmos para calcular todos los reductos más cortos [Susmaga, 1998; Lin and Yin, 2004; Zhou et al., 2009; Piza-Dávila et al., 2020], además, se han propuesto algunos que calculan solo aproximaciones de estos, por ejemplo el reportado en [Alwesabi et al., 2016], por lo que se hace necesario el desarrollo de algoritmos eficientes que calculen todos los reductos más cortos.

1.3 Objetivos

El **objetivo general** de esta investigación es:

Desarrollar un algoritmo para calcular todos los reductos más cortos; que sea más rápido que los algoritmos presentes en el estado del arte en algunos conjuntos de datos específicos.

Los **objetivos específicos** son:

1. Identificar un conjunto de características de la Matriz de Discernibilidad Binaria Simplificada que nos permita estimar el tamaño de los reductos más cortos.

2. Seleccionar un subconjunto de características de (1) que nos permita estimar el

tamaño de los reductos más cortos.

3. Desarrollar un algoritmo para calcular todos los reductos más cortos usando la estimación obtenida en (2).

1.4 Organización de la tesis

El resto de este documento se divide de la siguiente manera. En el Capítulo 2 se presentan los conceptos que se usarán a través de este documento. En el Capítulo 3 se muestran los algoritmos reportados en la literatura para calcular reductos más cortos mencionando sus ventajas y limitaciones. El Capítulo 4 contiene una explicación detallada del algoritmo propuesto en esta tesis. Los experimentos y resultados obtenidos con el algoritmo propuesto se presentan en el Capítulo 5. Finalmente las conclusiones y el trabajo futuro se exponen en el Capítulo 6.

Capítulo 2

Conceptos Básicos

La Teoría de Conjuntos Rugosos *(TCR)*, propuesta en 1982 por Zdzislaw Pawlak, se encuentra en un estado de constante desarrollo, enfocándose en la clasificación y análisis de información y conocimiento impreciso, incierto o incompleto. Esta teoría se considera uno de los primeros enfoques no estadísticos en el análisis de datos [Pawlak, 1982]. En este capítulo se presentan los conceptos teóricos de la teoría de conjuntos rugosos necesarios para comprender el trabajo realizado en esta tesis. Adicionalmente, se describe el concepto de contribución de atributos que es utilizado para podar el espacio de búsqueda de reductos.

2.1 Sistema de Decisión

La forma más común de representar los datos de un problema supervisado, en la TCR es a través de los Sistemas de Decisión *(SD)*. Un SD se define formalmente como:

Definición 2.1. *Un Sistema de Decisión es representado mediante un par* $T = (U, C \cup \{d\})$ *donde:*

$U = \{x_1, .., x_n\}$ *es un conjunto no vacío de objetos llamados universo.*

$C = \{c_1, .., c_k\}$ *es un conjunto no vacío de atributos, denominados atributos de condición.*

d *es un atributo, denominado atributo de decisión, tal que* $d \notin C$.

A cada atributo se le asocia una función $a : U \to V_a \quad para \quad a \in C \cup \{d\}$ *donde* V_a *es el conjunto de valores de* a *y* $a(x_i)$ *representa el valor que toma el atributo* a *en el objeto* x_i.

Un Sistema de Decisión es representado mediante una tabla en la cual las filas y las columnas representan objetos y atributos respectivamente de un conjunto de datos. En un problema de clasificación, el *atributo de decisión* determina a qué clase pertenece un objeto. En el SD de la Tabla 2.1, d es el *atributo de decisión*; éste es un sistema de dos clases ya que $V_d = \{0, 1\}$. Los *atributos de condición* no determinan absolutamente la clase, pero ayudan a decidir a qué clase pertenece un objeto. En la clasificación supervisada, los *atributos de condición* son la única información disponible para clasificar nuevos objetos; mientras que el *atributo de decisión* solo está disponible para objetos en el conjunto de entrenamiento.

Tabla 2.1: Sistema de Decisión.

	c_1	c_2	c_3	d
x_1	1	3	0	0
x_2	1	0	0	0
x_3	3	1	1	1
x_4	3	3	2	1
x_5	4	2	3	1
x_6	4	3	1	0
x_7	4	2	5	1

2.2 Relación de Indiscernibilidad

Para representar, analizar y manipular el conocimiento en un Sistema de Decisión *(SD)*, [Pawlak, 1982] define la relación de indiscernibilidad como sigue:

Definición 2.2. *Dado un Sistema de Decisión* $T = (U, C \cup \{d\})$ *y* $B \subseteq C$*, la relación, denotada por* $IND(B|d) \subseteq U \times U$ *y llamada relación de B-indiscernibilidad, se define como:*

$$IND(B|d) = \{(x, y) \in U \times U : \forall a \in B,\ [c(x) = c(y)] \vee [d(x) = d(y)]\ \} \tag{2.1}$$

Las instancias (objetos) x, y en $IND(B|d)$ son indiscernibles por los atributos de B dado el atributo de decisión d.

A partir de la relación de indiscernibilidad se puede analizar la consistencia del SD, ya que la relación de indiscernibilidad crea una partición del universo (U) en subconjuntos disjuntos de objetos [Pawlak, 1982]. Cuando cada subconjunto de objetos en la partición generada por la relación de indiscernibilidad sobre el conjunto de atributos de condición C pertenece a una única clase se dice que el SD es *consistente*, en otro caso es *inconsistente* [Pawlak, 1982]. En esta tesis nos enfocaremos en los sistemas de decisión consistentes.

2.3 Reducto y Core

Las relaciones de indiscernibilidad dividen a U en subconjuntos disjuntos. Tal partición del universo es denotado por $U/IND(B|d)$ [Suraj, 2004]. Dado un conjunto arbitrario de atributos $R \subseteq C$, la partición $U/IND(R|d)$ no es necesariamente la misma que la partición $U/IND(C|d)$. Un conjunto de atributos individualmente necesarios y conjuntamente suficientes para preservar la partición de $U/IND(C|d)$ se le denomina *reducto* [Pawlak, 1982].

Definición 2.3. *Dado un Sistema de Decisión $T = (U, C \cup \{d\})$, un subconjunto de atributos $R \subseteq C$ se le denomina reducto de T, si R satisface las siguientes condiciones:*

i. $U/IND(R|d) = U/IND(C|d)$;

ii. $\forall c \in R, U/IND(R - \{c\}|d) \neq U/IND(C|d)$.

Si R cumple la condición i se le denomina super-reducto de T, independientemente de cumplir con ii.

Dado un Sistema de Decisión, pueden existir varios reductos de diferentes longitudes. Si existen atributos que pertenecen a todos los reductos, estos atributos forman un conjunto denominado *core*.

Definición 2.4. *Dado un Sistema de Decisión* $T = (U, C \cup \{d\})$*, se denomina core de T, denotado como CORE(T), al subconjunto* $CORE(T) \subseteq C$ *tal que:* $\forall\ R \subseteq C$ *reducto de* T*,* $CORE(T) \subseteq R$*;*

2.4 Matriz de Discernibilidad

Como la TCR modela las diferencias de los objetos basadas en la noción de *discernibilidad*, comúnmente se construye una matriz llamada Matriz de Discernibilidad *(MD)* para representar las relaciones de discernibilidad entre los objetos de un sistema de decisión T. Esta matriz es una matriz cuadrada de tamaño n (el número de objetos) y se define como sigue:

Definición 2.5. *Dado un Sistema de Decisión* $T = (U, C \cup \{d\})$*. La matriz de discernibilidad está representada por una matriz simétrica* M *de dimensión* $|U| \times |U|$*, en la cual el elemento* $M_{v,w}$*, asociado a los objetos* x_v*,* x_w *de* U *se define como:*

$$M_{v,w} = \begin{cases} \{c \in C : c(x_v) \neq c(x_w)\} & si \;\; d(x_v) \neq d(x_w) \\ \emptyset & en\ otro\ caso \end{cases} \tag{2.2}$$

Donde $c(x_v)$ *representa el valor del atributo de condición* c *en el objeto* x_v*, y* $d(x_v)$ *representa el valor del atributo de decisión en el objeto* x_v*.*

Cada celda en la *matriz de discernibilidad* contiene aquellos atributos en los que

los dos objetos (el correspondiente a la columna y el correspondiente a la fila) tienen diferentes valores. Dos objetos son *discernibles* con respecto a un conjunto de atributos si dicho conjunto es un subconjunto de la celda correspondiente en la matriz de discernibilidad. La Tabla 2.2 muestra la matriz de discernibilidad para el SD de la Tabla 2.1, la cual se representa mediante una matriz triangular inferior, en la cual se han omitido $\emptyset$'s por claridad.

Tabla 2.2: Matriz de Discernibilidad del SD de Tabla 2.1.

$x \in U$	1	2	3	4	5	6	7
1							
2							
3	$\{c_1, c_2, c_3\}$	$\{c_1, c_2, c_3\}$					
4	$\{c_1, c_3\}$	$\{c_1, c_2, c_3\}$					
5	$\{c_1, c_2, c_3\}$	$\{c_1, c_2, c_3\}$					
6			$\{c_1, c_2\}$	$\{c_1, c_2\}$	$\{c_2, c_3\}$		
7	$\{c_1, c_2, c_3\}$	$\{c_1, c_2, c_3\}$				$\{c_2, c_3\}$	

2.5 Matriz de Discernibilidad Binaria

Existe otra representación muy utilizada para modelar las diferencias entre objetos basadas en la noción de discernibilidad, la cual, al igual que en la Definición 4 se representa a través de una matriz, en este caso, una matriz binaria, la cual contiene las comparaciones entre objetos que pertenecen a diferentes clases, es decir, que poseen diferentes valores en el atributo de decisión. En la Matriz de Discernibilidad Binaria *(MDB)*, las columnas son atributos de condición y las filas representan pares de objetos que pertenecen a diferentes clases. La MDB se define como sigue:

Definición 2.6. *Dado un Sistema de Decisión $T = (U, C \cup \{d\})$. La matriz de discernibilidad binaria (MDB) es una matriz M, en la cual el elemento $M_{i,j}$, de la fila i y la columna j, representa la comparación entre dos objetos x_v, x_w , tal que $d(x_v) \neq d(x_w)$,*

dado un atributo de condición $c_j \in C$, y se define como sigue:

$$M_{i,j} = \begin{cases} 1 & si\ c_j(x_v) \neq c_j(x_w) \\ 0 & en\ otro\ caso \end{cases} \tag{2.3}$$

Donde $c_j(x_v)$ representa el valor del atributo de condición c_j en el objeto x_v, y $d(x_v)$ representa el valor del atributo de decisión en el objeto x_v.

En la Tabla 2.3 se muestra la MDB del SD de la Tabla 2.1, que al igual que la Tabla 2.2 representa la discernibilidad entre los objetos en un sistema de decisión. Si observamos el conjunto de atributos $\{c_1, c_2, c_3\}$ en la Tabla 2.2, se puede observar que los objetos x_1 y x_3 son discernibles por este conjunto de atributos, al igual que en la Tabla 2.3 la fila uno, corresponde a la comparación entre x_1 y x_3, como en las columnas 1,2 y 3 el valor es 1 esto nos indica que cualquiera de los atributos en $\{c_1, c_2, c_3\}$ nos permite discernir entre x_1 y x_3.

Tabla 2.3: Matriz de Discernibilidad Binaria del SD de la Tabla 2.1.

	c_1	c_2	c_3
x_1, x_3	1	1	1
x_1, x_4	1	0	1
x_1, x_5	1	1	1
x_1, x_7	1	1	1
x_2, x_3	1	1	1
x_2, x_4	1	1	1
x_2, x_5	1	1	1
x_2, x_7	1	1	1
x_3, x_6	1	1	0
x_4, x_6	1	0	1
x_5, x_6	0	1	1
x_6, x_7	0	1	1

Dado que la matriz de discernibilidad binaria, modela la discernibilidad entre objetos, se puede usar para redefinir el concepto de reducto como sigue a continuación.

Definición 2.7. *Dada una matriz de discernibilidad binaria M, un subconjunto $R \subseteq C$ se le denomina reducto de M, si satisface las condiciones:*

i. $\forall$ *fila* i *en* M, $\exists c_j \in R$ *tal que* $M_{i,j} = 1$*;*

ii. $\forall c_j \in R$, $\exists$ *una fila* i *en* M *tal que* $\forall c_p \in (R - \{c_j\})$, $M_{i,p} = 0$.

Si R cumple la condición i se le denomina super-reducto de M, independientemente de cumplir con ii.

La primera condición especifica que para que un conjunto de atributos R sea un reducto, no puede existir ninguna fila en la matriz de discernibilidad binaria que contenga solamente ceros al considerar solo los atributos que pertenecen a R, es decir, cualquier par de objetos de diferente clase pueden discernirse con al menos un atributo en R. La segunda condición establece que si se elimina cualquiera de los atributos de un reducto esto produciría una fila de ceros, es decir, al eliminar cualquier atributo en R existe al menos un par de objetos de diferente clase que no pueden discernirse con los atributos restantes de R. Nótese que la Definición 2.7 es equivalente a la Definición 2.3.

2.6 Matriz de Discernibilidad Binaria Simplificada

La Matriz de Discernibilidad Binaria Simplificada *(MDBS)* es una versión más pequeña de la MDB obtenida como resultado de la eliminación de algunas de sus filas, a continuación se presentan algunas definiciones que se utilizan para eliminar dichas filas.

Definición 2.8. *Sean i, f filas en una matriz de discernibilidad binaria M, se dice que i es subfila de f si:*

i. $\forall$(*columna* j *de* M)$[(M_{i,j} = 1) \Rightarrow (M_{f,j} = 1)]$ *;*

ii. $\exists$(*una columna* q *de* M) *tal que* $M_{f,q} = 1$ *y* $M_{i,q} = 0$.

Las filas para las cuales no existan subfilas en la matriz de discernibilidad binaria se denominan *filas básicas*, y se definen como sigue:

Definición 2.9. *Sea f una fila en una MDB. La fila f es una* ***fila básica*** *si no existe fila alguna i que sea subfila de f.*

La matriz formada por las *filas básicas* de la matriz de discernibilidad binaria , sin repetir, se denomina *Matriz de Discernibilidad Binaria Simplificada (MDBS)*. Se ha demostrado en la literatura que esta nueva matriz contendrá los mismos reductos que la original [Yao and Zhao, 2009]. En la Tabla 2.4 se puede observar la MDBS para la MDB que se muestra en la Tabla 2.3.

Tabla 2.4: Matriz de Discernibilidad Binaria Simplificada del SD de la Tabla 2.1.

c_1	c_2	c_3
1	0	1
1	1	0
0	1	1

2.7 Contribución de un atributo

Dado un subconjunto de atributos, se dice que un atributo contribuye a dicho subconjunto para discernir más objetos si la unión entre el atributo y el subconjunto logra discernir más objetos que al considerar solamente al subconjunto original. Partiendo de la MDBS el concepto de contribución de un atributo se define como:

Definición 2.10. *Sea M una MDBS, $B \subseteq C$ un subconjunto de atributos y $c_j \in C$ un atributo, tal que $c_j \notin B$. Se dice que el atributo c_j contribuye a B si y solo si el número de filas de ceros, en la sub-matriz de M al considerar las columnas correspondientes a los atributos en $B \cup \{c_j\}$ es menor que la cantidad de filas de ceros al considerar las columnas correspondientes a los atributos en B. Lo cual es equivalente a decir que el*

atributo c_j contribuye a B si y solo si el número de filas con al menos un uno, en la sub-matriz de M al considerar las columnas correspondientes a los atributos en $B \cup \{c_j\}$ es mayor que la cantidad de filas con al menos un uno al considerar las columnas correspondientes a los atributos en B.

A partir de este concepto, en [Sanchez-Díaz and Lazo-Cortés, 2007] se planteó y probó la siguiente proposición.

Proposición 2.1. *Sea $B \subseteq C$ un subconjunto de atributos y $c_j \in C$ un atributo, tal que $c_j \notin B$. Si c_j no contribuye a B, entonces $B \cup \{c_j\}$ no formará parte de algún reducto.*

2.8 Síntesis

En este capítulo se proporcionaron los conceptos básicos de la Teoría de Conjuntos Rugosos necesarios para comprender el trabajo realizado en esta tesis. En particular los conceptos de reducto y super-reducto así como el concepto de contribución de atributos que es utilizado para podar el espacio de búsqueda de reductos. Conceptos en los cuales se sustenta la presente investigación.

Capítulo 3

Trabajo Relacionado

Como ya se mencionó, un reducto se define como un subconjunto de atributos que posee la misma capacidad de discernir objetos en un sistema de decisión que todo el conjunto de atributos condicionales y no contiene un subconjunto propio que cumpla dicha condición, ver Definición 2.3. Las técnicas utilizadas para el cálculo de los reductos se pueden clasificar en tres categorías: 1) aquellos que encuentran *super-reductos* (sí encuentran subconjuntos que tienen la misma capacidad de discernibilidad que todo el conjunto de atributos condicionales pero no necesariamente son irreducibles), 2) aquellos que encuentran *reductos* y 3) aquellos que encuentran los reductos de cardinalidad más pequeña *(reductos más cortos)*.

En la Figura 3.1 se muestra una taxonomía de los algoritmos en la literatura para el cálculo de reductos atendiendo a las diferentes categorías descritas con anterioridad, resaltando en el recuadro rojo los algoritmos en la literatura para el cálculo de los reductos más cortos, ya que en esta tesis nos centramos en estos algoritmos.

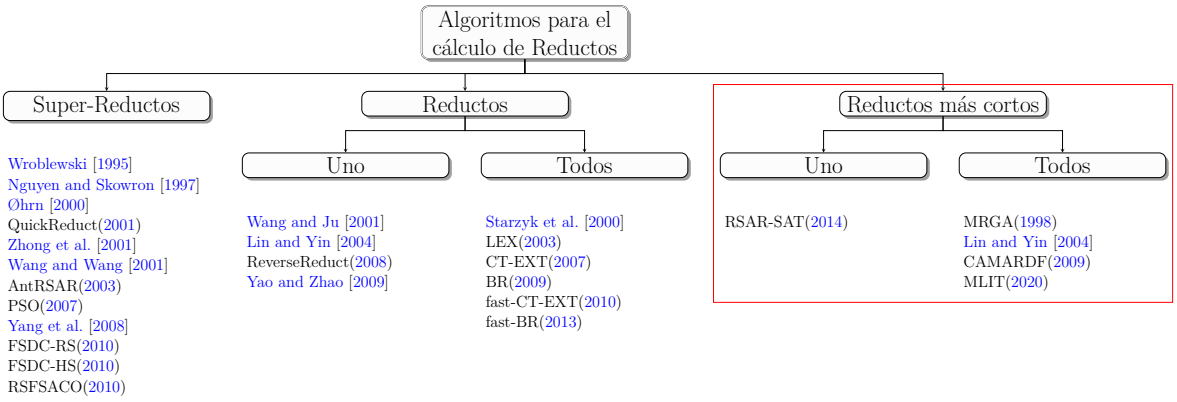


Figura 3.1: Taxonomía de los algoritmos para el cálculo de reductos

Como se puede observar en la Figura 3.1, existen varios algoritmos presentes en la literatura que calculan super-reductos o reductos y algunos que calculan reductos más cortos. Para algunas aplicaciones no son necesarios todos los reductos [Yu et al., 2009; Grzymala-Busse, 2005], en algunos estudios se ha evaluado la factibilidad de calcular todos los reductos más cortos (reductos de menor cardinalidad), en lugar de calcular todos los reductos o calcular super-reductos [Sil and Das, 2012]. Como se puede observar en la Figura 3.1 (recuadro rojo), existen muy pocos algoritmos para calcular todos los reductos más cortos. Por lo tanto, en este capítulo, se abordan los trabajos presentes en la literatura que calculan reductos más cortos, dado que en esta tesis se propone un algoritmo para calcular este tipo de reductos. A continuación, en la sección Sección 3.1, se describen cada uno de los algoritmos para el cálculo de los reductos más cortos y en la Sección 3.2 se realiza una discusión de las principales limitaciones de estos algoritmos.

3.1 Cálculo de reductos más cortos

Uno de los primeros trabajos reportados en la literatura para calcular todos los reductos más cortos se presentó en [Susmaga, 1998]. En este trabajo, se introducen dos algoritmos, uno para calcular los *K-reductos* (reductos de longitud menor o igual que K) y otro para calcular todos los reductos más cortos. Ambos algoritmos forman parte del Algoritmo de Generación de Reductos Modificado *(MRGA)*, que es la propuesta central del autor. En el algoritmo MRGA se introduce la aplicación de leyes de absorción sobre la función de discernibilidad, como una representación de la información de discernibilidad, esto permite reducir el tiempo de ejecución. Sin embargo, en este enfoque, cada candidato a reducto es evaluado buscando atributos superfluos. Ésta es una operación con un alto costo computacional, que reduce el rendimiento del algoritmo en algunos casos.

En [Lin and Yin, 2004], se propone un algoritmo para calcular todos los reductos más cortos. La estrategia principal que se sigue en [Lin and Yin, 2004] consiste en buscar heurísticamente un super-reducto pequeño que se usa para reducir el espacio de búsqueda, considerando sólo los subconjuntos de atributos con una cardinalidad igual o más baja. El principal inconveniente de este algoritmo es que se deben de verificar todas las posibles *s-subtablas* del conjuntos de datos. Una *s-subtabla* se define como una subtabla cuyo conjunto de atributos condicionales tiene tamaño s. En otras palabras, es una tabla de decisión con un subconjunto de atributos condicionales de tamaño s, más el atributo de decisión de la tabla original. Para buscar el super-reducto que se utiliza para limitar el espacio de búsqueda se sigue una heurística similar a la propuesta en [Chouchoulas and Shen, 2001], ya que se comienza con un subconjunto vacío R y se agregan, uno a la vez, los atributos que tienen mayor importancia (según el número de objetos discernibles por dicho atributo), este proceso termina cuando R tenga el mismo poder de discernibilidad que todo el conjunto de atributos condicionales. Otra desventaja de este algoritmo es que no utiliza alguna estrategia de poda y explora todas las posibles combinaciones de atributos para cada s-subtabla, lo que es muy costoso en la mayoría de los casos.

El algoritmo propuesto en [Zhou et al., 2009]($CAMARDF$) calcula todos los reductos más cortos en un Sistema de Decisión. En este algoritmo, el autor utiliza una versión reducida de la función de discernibilidad, la cual es construida a partir de la aplicación de las leyes de absorción sobre la función de discernibilidad [Rauszer and Skowron, 1992]. La estrategia de búsqueda utilizada en el algoritmo CAMARDF es primero en profundidad. En cada iteración, los atributos son ordenados de acuerdo a su importancia, para luego ser procesados en este orden. La importancia de un atributo

en el algoritmo CAMARDF está dada por el número de objetos discernibles al incluir dicho atributo en el candidato actual. Esta estrategia reduce el espacio de búsqueda, sin embargo, calcular la importancia de un atributo tiene un alto costo computacional. Otro aspecto a tener en consideración en el algoritmo CAMARDF es que los atributos que aparecen sólo una vez en la función de discernibilidad no son considerados en el proceso de búsqueda, pues estos atributos o bien pertenecen al core u otros atributos pueden ser considerados en su lugar [Zhou et al., 2009].

El algoritmo propuesto por Jensen et al. [2014] *(RSAR-SAT)* está enfocado al cálculo de un único reducto más corto en un Sistema de Decisión. El método introducido en este trabajo transforma el problema de encontrar un reducto utilizando la función de discernibilidad a un problema de satisfactibilidad *SAT* [Davis et al., 1962], es decir, dada una fórmula Booleana (típicamente en forma normal conjuntiva), el problema SAT consiste en la asignación de valores de verdad a los atributos de forma que la evaluación de la fórmula sea verdadera, o la determinación de que tal asignación no existe. Si luego de asignar los valores de verdad para todos los atributos que aparecen en las cláusulas la fórmula Booleana es satisfactible, entonces el conjunto de atributos con valores verdaderos forman un reducto más corto. El procedimiento que se sigue es similar al algoritmo *DPLL* propuesto por Davis Putnam Logemann Lovelandes en [Nieuwenhuis et al., 2005], en el cual los atributos son elegidos iterativamente y eliminados mediante la resolución de cada cláusula en la que que el atributo es verdadero con una cláusula en la que el atributo es falso. La función Booleana generada de esta manera siempre se satisface, ya que se considera el conjunto completo de atributos como una solución trivial.

Recientemente, un nuevo algoritmo que calcula todos los reductos más cortos fue propuesto por Piza-Dávila et al. [2020]. Este algoritmo denominado *MLIT* utiliza dos

estrategias para realizar la búsqueda de los reductos más cortos y decide cuál aplicar en función de la densidad de la matriz de discernibilidad binaria simplificada. La primera estrategia denominada *PFRC-MiLIT* se aplica si la densidad de la MDBS es menor que 0,3 y realiza una búsqueda de los reductos en amplitud, al contrario de la mayoría de los algoritmos discutidos con anterioridad que optan por una búsqueda en profundidad. La búsqueda en amplitud asegura que nunca se verificará un reducto que no sea de tamaño más corto. La segunda estrategia denominada *NC-MiLIT* se aplica si la densidad de la MDBS es mayor o igual a 0,3 y construye subconjuntos de atributos en orden ascendente, de tal forma, que nunca se generará un subconjunto de atributos de mayor tamaño hasta haber generado todos los subconjuntos de atributos del tamaño actual, lo cual asegura, que al encontrar un reducto, éste sea de menor tamaño. Además, en ambas estrategias se utiliza el concepto de contribución de un atributo para evaluar la condición de reducto en el menor número de subconjuntos posibles.

3.2 Discusión

Calcular la Matriz de Discernibilidad Binaria Simplificada *(MDBS)* del conjunto de datos original tiene una complejidad cuadrática, $O(|U|^2)$ respecto al número de objetos (filas) en el conjunto de datos [Yang et al., 2008], mientras que calcular todos los reductos tiene una complejidad exponencial, $O(2^{|C|})$ con respecto al número de atributos (columnas) [Rauszer and Skowron, 1992]. Además, en la mayoría de los conjuntos de datos, es mejor trabajar sobre la MDBS. Una de las principales desventajas que se notó en el estudio de los algoritmos para el cálculo de los reductos más cortos es que la mayoría de estos algoritmos no trabajan con la MDBS [Susmaga, 1998; Lin and Yin, 2004; Zhou et al., 2009; Jensen et al., 2014], sino que trabajan con la función de discernibilidad [Susmaga, 1998] y algunos con una versión simplificada de esta función

de discernibilidad [Zhou et al., 2009; Jensen et al., 2014], como resultado de aplicar las leyes de absorción sobre la función de discernibilidad [Rauszer and Skowron, 1992]. En [Zhou et al., 2009] se muestra que el tiempo principal para buscar los reductos más cortos en el algoritmo CAMARD se consume en construir la función de discernibilidad reducida. Por otra parte, en [Lin and Yin, 2004], las operaciones se realizan directamente sobre el Sistema de Decisión, lo cual es ineficiente en la mayoría de los conjuntos de datos y además, no se usa ninguna estrategia de poda.

Al contrario de los algoritmos presentados en [Susmaga, 1998; Lin and Yin, 2004; Zhou et al., 2009; Jensen et al., 2014], el algoritmo MLIT recientemente reportado en [Piza-Dávila et al., 2020] utiliza la Matriz de Discernibilidad Binaria Simplificada y propone algunos mecanismos de poda basados en el concepto de contribución de un atributo. Una de las principales limitaciones del algoritmo está dada por la estrategia PFRC-MiLIT, la cual se aplica si la densidad de la matriz de discernibilidad binaria simplificada es menor a 0.3, debido a que todas combinaciones de atributos del tamaño a chequear son almacenadas en memoria, por lo tanto el algoritmo MLIT puede ser inaplicable en conjuntos de datos con un número de atributos relativamente grande. Otra desventaja del algoritmo MLIT es que no se utiliza el core para podar el espacio de búsqueda, dado que si el core no es vacío, se puede llegar a reducir significativamente el número de candidatos a verificar [Ye and Chen, 2002].

En el algoritmo que se introduce en esta tesis, al igual que en el algoritmo MLIT, se utiliza la Matriz de Discernibilidad Binaria Simplificada. Pero, a diferencia del algoritmo MLIT, en el algoritmo propuesto se utiliza el core para podar el espacio de búsqueda y se introducen algunas estrategias de poda basadas en el concepto de contribución de un atributo y en un nuevo tipo de partición del conjunto de atributos. Otro aspecto

importante a considerar, es que, en el algoritmo propuesto, se parte de una idea similar a la presentada en [Lin and Yin, 2004], ya que primero, se aproxima el tamaño de los reductos más cortos, para luego determinar el tamaño de los reductos más cortos, pero en el algoritmo propuesto se utiliza un modelo de regresión polinómica para obtener la aproximación del tamaño de los reductos más cortos, al contrario del algoritmo propuesto en [Lin and Yin, 2004] que utiliza una búsqueda heurística de un super-reducto pequeño, lo cual es más costoso, además de que en el algoritmo de [Lin and Yin, 2004] no se utilizan estrategias de poda del espacio de búsqueda.

3.3 Síntesis

En este capítulo, se discutieron los trabajos más relevantes relacionados con la presente investigación. Una de las principales desventajas que se notó en el estudio de los algoritmos para el cálculo de los reductos más cortos es que la mayoría de estos algoritmos no trabajan con la Matriz de Discernibilidad Binaria Simplificada y otros utilizan representaciones complejas de los datos que están asociadas a operaciones computacionales ineficientes. De la revisión de la literatura se encontraron 4 algoritmos para calcular todos los reductos más cortos Figura 3.1, siendo el más reciente el reportado en [Piza-Dávila et al., 2020].

Capítulo 4

Algoritmo para el cálculo de los reductos más cortos

En este capítulo se introduce un algoritmo para calcular todos los reductos más cortos *(RMC)*. Inicialmente se describen las ideas principales en las cuales se basa el algoritmo propuesto, a continuación de la Sección 4.1 a la Sección 4.3 se describe a detalle cada uno de los pasos que conforman al algoritmo. Finalmente en la Sección 4.4 se presenta el análisis de complejidad del algoritmo propuesto.

La estrategia más simple para realizar el cálculo de todos los reductos más cortos consiste en verificar la condición de reducto para todos los subconjuntos de atributos del conjunto completo de atributos y seleccionar aquellos subconjuntos de menor tamaño para los cuales se cumple la condición de reducto. Esta estrategia se puede mejorar si los subconjuntos de atributos se generan en cierto orden, por ejemplo generando y evaluando todos los posibles subconjuntos para cada posible tamaño K=*1,...,n*, siendo n el número de atributos en el sistema de decisión. Para esto existen al menos dos formas en las cuales se pueden generar los subconjuntos de atributos. Una opción es verificar los subconjuntos de atributos de forma incremental, partiendo de los subconjuntos de atributos de tamaño 1 e incrementando de uno en uno el tamaño de los subconjuntos cuando no se encuentren super-reductos del tamaño actual; hasta llegar a un tamaño K para el cual se encuentre al menos un super-reducto y entonces encontrar todos los super-reductos de tamaño K. Si no se encuentran super-reductos, este proceso termina al llegar al conjunto completo de atributos. De forma inversa, podemos verificar subconjuntos de atributos de forma decremental, partiendo del conjunto total de atributos y

decrementando de uno en uno el tamaño de estos subconjuntos, hasta llegar a un tamaño *K-1*, para el cual no hay super-reductos y entonces buscar todos los super-reductos de tamaño *K*. No obstante, independiente de la estrategia utilizada (incremental o decremental), el número de subconjuntos de atributos que se deben verificar para determinar el tamaño de los reductos más cortos en conjuntos de datos grandes puede ser excesivo, por lo que es necesario buscar alguna manera de verificar la condición de super-reducto en la menor cantidad posible de subconjuntos de atributos (podar el espacio de búsqueda). Para esto, conocer el tamaño de los reductos más cortos, o tener una buena aproximación del mismo, puede ayudar a dirigir la búsqueda de forma eficiente y, de este modo, reducir el número de subconjuntos de atributos evaluados.

El algoritmo que se introduce parte de la idea de que el cálculo de los reductos más cortos se puede realizar de forma más eficiente si se conoce a priori el tamaño de éstos, ya que al conocer el tamaño, el problema se reduciría a encontrar todos los super-reductos de dicho tamaño. Con esto en mente, de forma general, el algoritmo propuesto sigue la idea que se muestra gráficamente en la Figura 4.1, donde primero se realiza una estimación del tamaño de los reductos más cortos, luego partiendo de esta estimación se determina el tamaño real de los mismos. Finalmente, conociendo el tamaño real de los reductos más cortos se realiza la búsqueda de todos los super-reductos de dicho tamaño.

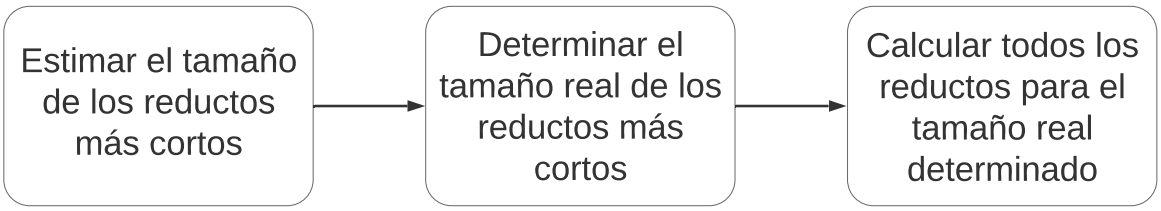


Figura 4.1: Estrategia del algoritmo propuesto.

Un aspecto importante a tener en cuenta es que, a pesar de conocer el tamaño de los reductos más cortos, el número de subconjuntos a evaluar puede seguir siendo excesivo, pues se tendría que evaluar cada posible subconjunto del tamaño dado. En esta tesis, en la Sección 4.2 se introducen algunas estrategias que permiten eliminar subconjuntos de atributos que no pertenecen a algún reducto, basadas en los conceptos de core, ver Definición 2.4, y contribución de un atributo, Proposición 2.1.

4.1 Estimación del tamaño de los reductos más cortos

El tamaño de los reductos más cortos *(RMC)* puede variar considerablemente dependiendo de las características de la Matriz de Discernibilidad Binaria Simplificada *(MDBS)*, una de las características que influye es el número de columnas de la MDBS, ya que el tamaño de los reductos estará limitado por dicho valor. El número de filas de la MDBS es otra característica importante que influye en el tamaño de los RMC, ya que en matrices con un número de filas menor al número de columnas, el tamaño de los reductos estará limitado por el número de filas. Otra característica que influye en el tamaño de los RMC es la densidad de unos en la MDBS, ya que, en muchos casos, una densidad alta conlleva a RMC de tamaño pequeño y matrices con densidad baja tienen asociados RMC de mayor tamaño [Rodríguez-Diez et al., 2018]. No obstante, estas características no son suficientes para estimar adecuadamente el tamaño de los reductos más cortos, por lo que en esta tesis utilizamos otras características, como la media y la desviación estándar de la cantidad de unos por filas y por columnas, las cuales proporcionan información sobre la dispersión de los unos en la MDBS.

En la Tabla 4.1 se muestran las características antes mencionadas para algunas matrices de discernibilidad binaria simplificada, donde N_F es el número de filas, N_C es el número de columnas, $DENS$ es la densidad de unos en la matriz, MED_C es la media del número de unos por columna, MED_F es la media del número de unos por fila, $STDV_C$ la desviación estándar del número de unos por columna, $STDV_F$ la desviación estándar del número de unos por fila, T_{RMC} es el tamaño de los RMC y CCP es el Coeficiente de Correlación de Pearson de cada característica con el tamaño de los reductos más cortos. En la Tabla 4.1 se puede observar que existe una correlación inversa entre las características de la matriz de discernibilidad binaria simplificada y el tamaño de los reductos más cortos, en la misma, se puede apreciar que la covariación extrema es más clara para las características de densidad, media de unos por filas y desviación estándar de unos por filas, lo cual indica, que estas características son las de mayor correlación con el tamaño de los reductos más cortos.

Nombre de la MDBS	N_F	N_C	$DENS$	MED_C	MED_F	$STDV_C$	$STDV_F$	T_{RMC}
biodeg	40	41	0.083	1.0	3.0	3.732	1.815	13
connect	406	42	0.047	22.0	2.0	11.496	0.000	24
dermatology	1103	34	0.344	356.0	12.0	133.419	3.581	6
lung-cancer	237	56	0.458	113.0	26.0	40.725	5.682	4
MED	26	36	0.030	1.0	1.0	0.460	0.423	25
sponge	68	45	0.382	26.0	16.0	15.903	5.979	1
student-por	8158	32	0.413	3301.0	13.0	1247.249	2.038	7
flags	390	29	0.353	123.0	10.0	73.441	2.320	1
kr-vs-kp	29	36	0.029	1.0	1.0	0.345	0.253	29
sponge	68	45	0.382	26.0	16.0	15.903	5.979	1
CCP	-0.160	-0.178	-0.918	-0.173	-0.808	-0.182	-0.840	1

Tabla 4.1: Valores de las características extraídas para algunas $MDBS$.

Para estimar el tamaño de los reductos más cortos en este trabajo de tesis se usa el modelo de ***regresión lineal múltiple*** [Granados, 2016]. En el modelo de regresión lineal múltiple se supone que la función de regresión que relaciona la variable dependiente

Y con las variables independientes $X_1, X_2, .., X_p$ es lineal, de la forma:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon \tag{4.1}$$

Una de las condiciones que exige la regresión lineal múltiple es que la relación entre la variable dependiente y las variables independientes sea lineal [Granados, 2016]. Para analizar dicha relación se utilizó un conjunto de datos donde se recopilaron las características de las matrices de discernibilidad binaria simplificada, ver Tabla 4.1, obtenidas a partir de conjuntos de datos del repositorio *UCI* [Dua and Graff, 2019]. Al analizar la relación entre el tamaño de los reductos más cortos (variable dependiente) y las características extraídas, se pudo observar que no existe una relación lineal entre la variable dependiente y las variables independientes, por lo que se realizó una transformación en las variables independientes para lograr una cierta relación lineal. Uno de los procedimientos más usuales de trasformación de las variables independientes para variables cuantitativas consisten en transformar cada variable independiente en polinomios, de forma que se consiga una cierta linealidad de la relación [Granados, 2016], es decir, además de introducir la variable x como variable independiente se introduce $x^2; x^3$,etc. En la mayoría de los casos basta con incluir una potencia cuadrática [Granados, 2016], por lo que reescribiendo la Ecuación 4.1 utilizando potencias cuadráticas obtenemos la siguiente ecuación:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2 + ... + \beta_{2p-1} X_p + \beta_{2p} X_p^2 + \epsilon \tag{4.2}$$

La Ecuación 4.2 es utilizada en el modelo de regresión polinómica, el cual nos permitirá relacionar el tamaño de los reductos más cortos con las características extraídas

de la MDBS, de modo que dadas las características de la MDBS este modelo permita obtener una estimación del tamaño de los reductos más cortos; como se mostrará en la sección de experimentos.

4.2 Determinación del tamaño de los reductos más cortos

Para determinar el tamaño real de los reductos más cortos se parte de una estimación del tamaño de estos reductos obtenida como se explicó en la sección anterior. Esta estimación se va corrigiendo hasta encontrar el tamaño real de los reductos más cortos. Antes de introducir el procedimiento que se utiliza para determinar el tamaño real de los reductos más cortos es necesario introducir algunos aspectos que permitirán establecer una primera cota del mismo.

4.2.1 Cálculo del core

Para establecer una *cota inferior* del tamaño real de los reductos más cortos se utiliza el core, ya que el core está formado por aquellos atributos imprescindibles para mantener la discernibilidad del sistema de decisión, es decir, si se elimina cualquier atributo perteneciente al core se pierde la capacidad de discernibilidad del sistema de decisión; esto quiere decir que los atributos del core pertenecen a todos los reductos, por lo tanto, si el core no es vacío, se toma el número de atributos que pertenecen al core como cota inferior del tamaño de los reductos más cortos.

Para calcular el core se parte de la Definición 2.4, la cual plantea que el core está constituido por el conjunto de atributos que pertenecen a todos los reductos. El core puede ser calculado rápidamente sin necesidad de conocer de antemano todos los re-

ductos, a continuación se muestran las ideas que se utilizan para realizar el cálculo del core de forma rápida.

En una MDBS cada fila representa la comparación entre los atributos de dos objetos pertenecientes a diferentes clases y las columnas representan a los atributos, ver Tabla 2.3, si existe una fila en la MDBS en la cual todos los valores de sus columnas fueran cero, con excepción de una, el atributo que corresponde a la columna con valor 1 sería un atributo indispensable para mantener la discernibilidad, por lo tanto, dicho atributo tiene que pertenecer al core. Todos los atributos (columnas en la MDBS) que cumplan esta condición forman parte del core.

Siguiendo la idea anterior, el cálculo del core se puede reducir a la búsqueda de las filas en la MDBS con un solo 1. En el Algoritmo 4.1 se muestra el procedimiento que se utiliza para generar el core siguiendo la idea anterior. Si aplicamos dicho algoritmo a la MDBS de la Tabla 4.2 se puede observar que en este caso el core está formado solo por el atributo c_5, ya que la fila 3 solo tiene valor 1 en la columna 5; y es la única fila con un solo 1.

Algoritmo 4.1: Generar core

```
   Entrada: Una MDBS T
   Salida  : core de T
1  core ← ∅
2  mientras queden filas no visitadas en T hacer
3  |  seleccionar nueva fila f en T y marcarla como visitada
4  |  si f contiene un solo 1 entonces
5  |  |  adicionar el atributo correspondiente a la columna con valor 1 al core
6  |  fin
7  fin
8  devolver core
```

c_1	c_2	c_3	c_4	c_5
0	1	0	1	0
1	0	1	0	0
0	0	0	0	1
0	1	1	0	0

Tabla 4.2: Matriz de Discernibilidad Binaria Simplificada.

Para fijar una cota superior del tamaño de los reductos más cortos se realiza una partición del conjunto de atributos, dicha partición se utiliza en esta tesis como uno de los mecanismo de poda del espacio de búsqueda y se explica a continuación.

4.2.2 Partición del conjunto de atributos

La idea principal es realizar una partición de los atributos en un Sistema de Decisión *(SD)* de forma tal que exista un atributo en cada subconjunto resultante de dicha partición, que logre discernir a todos los objetos discernidos por cada uno de los restantes atributos del subconjunto, es decir, que tenga la misma o mayor capacidad de discernibilidad que todos los atributos del subconjunto al cual pertenece. Es decir, dada una MDBS M, se buscan subconjuntos de atributos, en donde, para cada subconjunto $S \subseteq C$, existe un atributo $c_q \in S$ tal qué para todo atributo $c_p \in S$, $c_q \neq c_p$: en las filas de M en las que en la columna p hay un uno, en la columna q también, es decir, el conjunto de filas con uno en la columna p es un subconjunto de las filas con uno en la columna q. Si recordamos la definición de contribución de un atributo a un subconjunto de atributos Definición 2.10, se puede notar que, en el subconjunto S, cada atributo c_p contribuye a cualquier subconjunto de atributos a lo sumo tanto como el atributo c_q, por tanto, si el atributo c_q contribuye a un subconjunto de atributos, pero no lo suficiente para formar un super-reducto, ningún atributo c_p formará un super-reducto con dicho subconjunto, esto se debe a que todo atributo c_p contribuye a lo sumo tanto

como c_q, pues el conjunto de filas que tienen al menos un uno en la columna p es un subconjunto de las filas que tienen un uno en la columna q.

Para realizar una partición del conjunto de atributos que cumpla con la propiedad propuesta se parte de una idea similar a la presentada en la Definición 2.8 y la Definición 2.9 las cuales permiten eliminar filas redundantes en la matriz de discernibilidad binaria. En el caso que nos ocupa se desea realizar una partición del conjunto de atributos según la capacidad de discernibilidad de los atributos, para este fin, a continuación se introducen los conceptos de *subcolumna* y *supercolumna*.

Definición 4.1. *Sea M una $MDBS$, p y q columnas en M, se dice que p es subcolumna de q si:*

$$\forall(fila\ i\ en\ M)[(M_{i,p} = 1) \Rightarrow (M_{i,q} = 1)] \tag{4.3}$$

adicionalmente si p es subcolumna de q se dice que q es supercolumna de p.

La Definición 4.1 establece una relación entre dos columnas de una MDBS, en la cual una columna p es *subcolumna* de otra columna q, si en las filas en las que p tiene un uno, q también lo tiene.

El Algoritmo 4.2 propuesto en esta tesis utiliza la Definición 4.1 para realizar una partición del conjunto de atributos de condición de un sistema de decisión construyendo subconjuntos de atributos en los cuales, la columna en la MDBS correspondiente al primer atributo de cada subconjunto será supercolumna de las columnas correspondientes al resto de los atributos de dicho subconjunto. La idea principal del algoritmo es crear los subconjuntos de atributos que forman la partición de los atributos iterativamente hasta que todas los atributos en C pertenezcan a algún subconjunto. En cada iteración se crea un nuevo subconjunto S a partir de los atributos que no han sido asignados con

anterioridad a ningún subconjunto, adicionando el primer atributo sin asignar a S y tomando el mismo como primer atributo en S, denominaremos este atributo como c_q. A continuación para cada atributo c_p no asignado a ningún subconjunto, si la columna asociada a c_q es supercolumna de p se adiciona c_p al subconjunto S y si la columna asociada a c_q es subcolumna de p se adiciona c_p al subconjunto S y se toma c_p como primer atributo en S, es decir se toma $c_q = c_p$ y se repite el procedimiento anterior con los atributos restantes no asignados a algún subconjunto creado. Cabe destacar que dicha partición no es única ya que ésta dependerá del orden en que se procesan las columnas en la MDBS, por ejemplo, si C=$\{c_1, c_2, c_3\}$ y la columna 1 es subcolumna de la columna 2 y 3, pero la columna 2 no es subcolumna de la columna 3 ni viceversa. Entonces, si procesamos las columnas en la MDBS de izquierda a derecha obtenemos la siguiente partición: $[\{c_2, c_1\}, \{c_3\}]$; pero si, por el contrario, procesamos las columnas de derecha a izquierda obtenemos: $[\{c_3, c_1\}, \{c_2\}]$. En el Algoritmo 4.2 se muestra el pseudocódigo del procedimiento descrito con anterioridad, en el cual, el procesamiento de las columnas en la MDBS se realiza de izquierda a derecha.

A continuación en la Tabla 4.3 se muestra un ejemplo de ejecución del Algoritmo 4.2 sobre la MDBS de la Tabla 4.2. La columna Relación representa la evaluación de la condición de subcolumna y supercolumna.

La partición generada por el Algoritmo 4.2 se utilizará como un mecanismo de poda del espacio de búsqueda, ya que en un inicio, cada candidato a super-reducto se generará a partir del subconjunto de atributos formado por el primer atributo de cada subconjunto en la partición del conjunto de atributos, este subconjunto se denotará como C'. La siguiente proposición constituye la base para realizar la poda del espacio de búsqueda, en el algoritmo que se introduce en este capítulo para el cálculo de todos

Algoritmo 4.2: Partición del conjunto de atributos

Entrada: Una MDBS M
Salida : Una lista de subconjuntos de atributos

```
1  L_S; lista que almacena los subconjuntos de atributos generados, la cual representa una
     partición del conjunto general de atributos
2  L_C; lista de todos los atributos en M
3  mientras L_C no esté vacía hacer
4      eliminar primer atributo en L_C y almacenarlo en c_q
5      S ← {c_q}
6      para todo atributo c_p en L_C hacer
7          si q es supercolumna de p entonces
8              adicionar c_p al final de S
9              eliminar c_p de L_C
10         fin
11         en otro caso
12             si q es subcolumna de p entonces
13                 adicionar c_p al principio de S
14                 c_q ← c_p
15                 eliminar c_p de L_C
16             fin
17         fin
18     fin
19     adicionar S a L_S;
20 fin
21 devolver L_S
```

L_S	Iteración	S	L_C	c_q	c_p	Relación	Comentarios
[]	1	$\{c_1\}$	$[c_2, c_3, c_4, c_5]$	c_1	c_2	ninguna	$c_p = c_3$
[]	2	$\{c_1\}$	$[c_2, c_3, c_4, c_5]$	c_1	c_3	subcolumna	eliminar el atributo c_3 de L_C, $c_q = c_3$, $S = \{c_3\} \cup S$, $c_p = c_4$
[]	3	$\{c_3, c_1\}$	$[c_2, c_4, c_5]$	c_3	c_4	ninguna	$c_p = c_5$
[]	4	$\{c_3, c_1\}$	$[c_2, c_4, c_5]$	c_3	c_5	ninguna	adicionar S a L_S, eliminar el atributo c_2 de L_C
$[\{c_3, c_1\}]$	1	$\{c_2\}$	$[c_4, c_5]$	c_2	c_4	supercolumna	eliminar el atributo c_4 de L_C, $S = S \cup \{c_4\}$, $s_p = c_5$
$[\{c_3, c_1\}]$	2	$\{c_2, c_4\}$	$[c_5]$	c_2	c_5	ninguna	adicionar S a L_S, eliminar el atributo c_5 de L_C
$[\{c_3, c_1\}, \{c_2, c_4\}]$	1	$\{c_5\}$	[]	c_5	-	-	adicionar S a L_S
$L_S = [\{c_3, c_1\}, \{c_2, c_4\}, \{c_5\}]$							

Tabla 4.3: Ejemplo de ejecución del Algoritmo 4.2, usando la MDBS de la Tabla 4.2.

los reductos más cortos.

Proposición 4.1. *Sea C el conjunto de todos los atributos correspondientes a las columnas en una MDBS y S un subconjunto de atributos en la partición del conjunto C,*

generada por el Algoritmo 4.2, tal que:

$$\exists c_q \in S \mid [\forall c_p \in S - \{c_q\}] \; ; q \; es \; supercolumna \; de \; p \tag{4.4}$$

Si c_q no contribuye a un subconjunto de atributos $R \subseteq C$, entonces, ningún otro atributo c_p en S contribuye a R.

Demostración

Sea M una MDBS, $C = \{c_1, c_2, c_3, ..., c_n\}$ el conjunto de atributos y $S = \{c_q, c_p\}$ un subconjunto de atributos de la partición del conjunto de atributos C obtenidos con el Algoritmo 4.2, para el cual se cumple que la columna q es supercolumna de la columna p.

Dado un subconjunto de atributos $R \subseteq C$. Sea F el conjunto formado por las filas que tienen cero en las columnas correspondientes a los atributos en R.

Primero, si F es vacío, R es un super-reducto, por lo tanto, ningún atributo en $C - R$ va a contribuir a R, entonces, se cumple la Proposición 4.1.

Luego, si F no es vacío, primero supongamos que el atributo c_q no contribuye a R, lo cual implica que para toda fila $i \in F$, $M_{i,q} = 0$. Ahora supongamos que el atributo c_p contribuye a R, esto implica que existe una fila $i \in F$, tal que, $M_{i,p} = 1$, por Definición 2.10, luego por definición de supercolumna $M_{i,p} = 1 \Rightarrow M_{i,q} = 1$ lo cual es una contradicción. Por lo que podemos concluir que si c_q no contribuye a un subconjunto R, ningún atributo $c_p \in S - \{c_q\}$ contribuye a R.

Recordando la Proposición 2.1, si un atributo $c_q \in C$ no contribuye a un subconjunto de atributos $R \subseteq C - \{c_q\}$, entonces $R \cup \{c_q\}$ no formará parte de algún reducto. Luego por Proposición 4.1 el siguiente resultado es inmediato.

Corolario 4.1. *Sea M una MDBS, $B \subseteq C$ un subconjunto de atributos y S un subconjunto de la partición del conjunto de atributos generada por el Algoritmo 4.2, en donde el atributo $c_q \in S$ satisface la Ecuación 4.4. Se cumple que:*
Si c_q no contribuye a B, entonces, para todo $c_p \in S - \{c_q\}$, $B \cup \{c_p\}$ no formará parte de algún reducto.

El Corolario 4.1 se utiliza para realizar una poda del espacio de búsqueda, ya que si un atributo c_q en un subconjunto de atributos S de la partición generada por el Algoritmo 4.2 satisface la Ecuación 4.4 y no contribuye a un subconjunto de atributos R, no será necesario verificar si los restantes atributos $c_p \in S$ contribuyen a R. No obstante, si el atributo c_q contribuye a R se tiene que verificar si $R \cup \{c_q\}$ es un super-reducto. La siguiente proposición se utiliza para verificar la condición de super-reducto en el menor número de subconjuntos posible.

Proposición 4.2. *Sea C el conjunto de todos los atributos correspondientes a las columnas en una MDBS y S un subconjunto de atributos en la partición del conjunto de atributos C generada por el Algoritmo 4.2, y el atributo c_q es el primer atributo en S, entonces se cumple que:*
Dado un subconjunto de atributos $R \subseteq C$, si $R \cup \{c_q\}$ no es un super-reducto, entonces, para todo atributo $c_p \in S - \{c_q\}$, $R \cup \{c_p\}$ tampoco es un super-reducto.

Demostración

Sea M una MDBS, $C = \{c_1, c_2, c_3, ..., c_n\}$ el conjunto de atributos y $S = \{c_q, c_p\}$ un subconjunto de atributos de la partición de los atributos en M según el Algoritmo 4.2, para el cual se cumple que la columna q es supercolumna de la columna p.

Dado un subconjunto de atributos $R \subseteq C$. Sea F el conjunto formado por las filas que tienen cero en las columnas correspondientes a los atributos en R.

Primero, si F es vacío, R es un super-reducto, por lo tanto, para todo atributo $c \in C - R$, $R \cup \{c\}$ es un super-reducto, entonces, se cumple la Proposición 4.2.

Luego si F no es vacío, primero supongamos que $R \cup \{c_q\}$ no es un super-reducto, lo cual implica que existe una fila $i \in F$, tal que, $M_{i,q} = 0$. Ahora supongamos que $R \cup \{c_p\}$ es un super-reducto, esto implica que para toda fila $i \in F$, se cumple que, $M_{i,p} = 1$, por Definición 2.7, luego por definición de supercolumna $M_{i,p} = 1 \Rightarrow M_{i,q} = 1$ lo cual es una contradicción. Por lo que podemos concluir que si $\{c_q\} \cup R$ no es un super-reducto, ningún atributo $c_p \in S - \{c_q\}$ en unión con R formará un super-reducto.

La siguiente proposición constituye la base para definir una cota superior del tamaño de los reductos más cortos.

Proposición 4.3. *Sea C el conjunto de todos los atributos correspondientes a las columnas en una MDBS y C' el subconjunto de atributos formado por el primer atributo de cada uno de los subconjuntos en la partición del conjunto de atributos C generada por el Algoritmo 4.2, entonces C' es un super-reducto.*

Demostración

Siendo consistente el sistema de decisión, sabemos que C es un super-reducto. Supongamos que C' no es un super-reducto, lo cual significa que C' puede ser completado para formar un super-reducto. Se requiere, para empezar, un atributo $c_p \in C - C'$ que contribuye a C'. Pero, c_p pertenece a algún subconjunto de atributos de la partición, por tanto existe un $c_q \in C'$ para el cual 'q es supercolumna de p', entonces c_p no puede contribuir a C'. Entonces podemos concluir C' es un super-

reducto.

Como resultado de lo planteado en la Proposición 4.3, el tamaño de los reductos estará limitado por el número de subconjuntos que inducen la partición del conjunto de atributos producida por el Algoritmo 4.2, ya que C' siempre es un super-reducto. En el siguiente corolario se muestra dicho resultado.

Corolario 4.2. *Sea M una $MDBS$ y N es el número de subconjuntos de atributos resultantes de realizar una partición del conjunto de atributos en M utilizando el Algoritmo 4.2, para la cual se cumple la Proposición 4.1, entonces el tamaño de los reductos será a lo sumo N.*

Discutidas las ideas anteriores, a partir de la estimación inicial del tamaño de los reductos más cortos, utilizando las cotas definidas con anterioridad, se sigue la estrategia que se muestra en la Figura 4.2. De forma general, si la estimación del tamaño de los reductos más cortos es menor que el número de atributos en el core, se fija como cota inferior del tamaño de los reductos más cortos el tamaño del core, en caso contrario, si la estimación del tamaño de los reductos más cortos es mayor al número de subconjuntos en la partición generada por el Algoritmo 4.2 se establece como cota superior el número de subconjuntos N en la partición generada por el Algoritmo 4.2.

La estrategia propuesta en esta tesis para determinar el tamaño de los reductos más cortos se puede observar en la Figura 4.3, inicialmente se parte de una estimación K del tamaño de los reductos más cortos. Luego se realiza una corrección de la estimación inicial mediante las cotas definidas con anterioridad, ver Figura 4.2. A continuación se verifica si existen super-reductos de tamaño K; si existen super-reductos de tamaño K se verifica si existen super-reductos de tamaño $K-1$, para ir corrigiendo en pasos

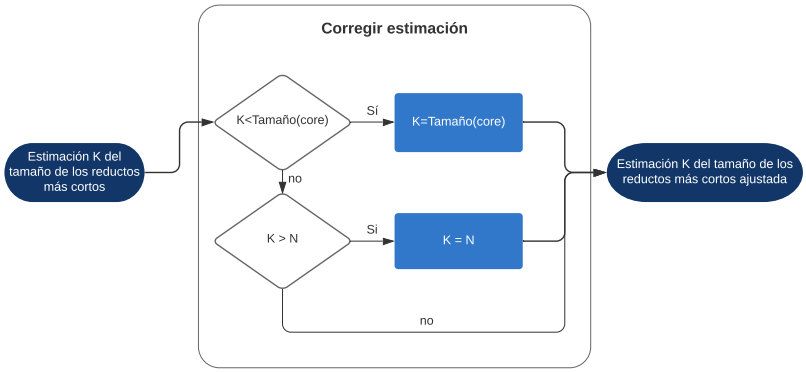


Figura 4.2: Corrección de la estimación del tamaño de los reductos más cortos.

sucesivos el tamaño de los super-reductos (estrategia decremental) hasta encontrar un tamaño para el cual no existan super-reductos y entonces el tamaño anterior será el de los reductos más cortos. En caso de que no exista un super-reducto de tamaño K se procede a verificar si existen super-reductos de tamaño mayor hasta encontrar el primer tamaño para el cual existan super-reductos (estrategia incremental), el cual corresponderá al tamaño de los reductos más cortos.

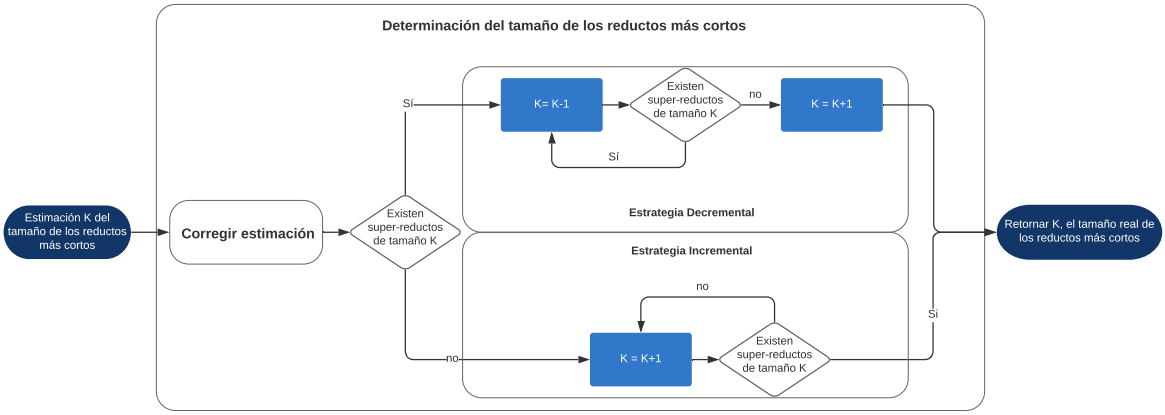


Figura 4.3: Estrategia utilizada para determinar el tamaño real de los reductos más cortos.

El problema de determinar el tamaño de los reductos más cortos se reduce a verificar en cada momento por la existencia de super-reductos para un tamaño determinado. Para

verificar por la existencia de super-reductos de tamaño K se debe evaluar la condición de super-reducto para cada posible candidato de dicho tamaño, ver Definición 2.3 *condición (i)*, sin embargo, dependiendo del número total de atributos y del tamaño a evaluar K, pueden ser demasiadas combinaciones, por lo que es necesario aplicar alguna estrategia de poda que permita eliminar candidatos a super-reductos. A continuación se introducen las estrategias que se siguen en esta tesis para eliminar candidatos a super-reductos.

4.2.3 Poda del espacio de búsqueda

Para realizar una primera poda del espacio de búsqueda se utiliza el core ya que el core está formado por aquellos atributos imprescindibles para mantener la discernibilidad del sistema de decisión, por lo tanto, si el core es un super-reducto, será el único reducto en la Matriz de Discernibilidad Binaria Simplificada *(MDBS)*, en caso de no ser un super-reducto, cada candidato a super-reducto se formará a partir de la unión del core con otros atributos que no están en el core, cabe destacar que en algunos casos el core puede ser vacío.

En caso de que el core sea vacío se utiliza el concepto de *conjunto mínimo de atributos* [Sanchez-Díaz and Lazo-Cortés, 2007] para podar el espacio de búsqueda.

Definición 4.2. *Sea M una MDBS, C el conjunto de atributos en M, y i la primera fila con menor número de unos en M. El* ***conjunto mínimo de atributos (CMA)*** *se define como:*

$$CMA = \{c_j \in C \mid M_{i,j} = 1 \} \tag{4.5}$$

El conjunto mínimo de atributos está formado por los atributos correspondientes a las columnas que en la fila con menor cantidad de unos tienen un uno. Este conjunto de atributos nos permite eliminar algunos candidatos a super-reductos, ya que cada

super-reducto tiene que contener al menos un atributo perteneciente al conjunto mínimo de atributos, pues según la definición de super-reducto; no puede existir alguna fila de ceros en la MDBS considerando solo los atributos que pertenecen al super-reducto.

De forma general, si el core no es vacío, se consideran como candidatos a super-reductos solo los subconjuntos de atributos formados a partir de la unión del core con otros atributos que no están en el core, en caso de que el core sea vacío se consideran como candidatos a super-reductos los subconjuntos de atributos que contengan al menos un atributo del conjunto mínimo de atributos, de forma tal que los atributos pertenecientes al CMA en el candidato a super-reducto sean siempre los primeros atributos en el candidato. Para determinar el tamaño de los reductos más cortos solo se considerarán como candidatos a super-reductos los subconjuntos de atributos $R \subseteq C'$, en donde C' es el conjunto formado por el primer atributo de cada uno de los subconjuntos en la partición generada por el Algoritmo 4.2, ya que si ningún subconjunto de atributos $R \subseteq C'$ de tamaño K es un super-reducto entonces ningún $R' \subseteq C$ de tamaño K será un super-reducto, por la Proposición 4.2.

Ejemplo 4.1. *Partiendo del resultado que se muestra en la Tabla 4.3, en donde* $L_S = [\{c_3, c_1\}, \{c_2, c_4\}, \{c_5\}]$*, cada candidato a super-reducto tiene que contener el core* $\{c_5\}$*, y ser subconjuntos del conjunto de atributos* C'*, formado a partir del primer atributo cada uno de los subconjuntos en la partición generada por el Algoritmo 4.2 que se denota como* L_S*, es decir,* $C' = \{c_3, c_2, c_5\}$*, los posibles candidatos a super-reductos de tamaño 2 son* $\{c_5, c_3\}$ *y* $\{c_5, c_2\}$*.*

A pesar de que las estrategias anteriores eliminan algunos candidatos a super-reductos, verificar si existen super-reductos para un tamaño K determinado puede seguir siendo un proceso costoso, por lo que es necesario otro mecanismo para podar

aún más el espacio de búsqueda. A continuación se presenta la idea que se utiliza para reducir aún más el espacio de búsqueda.

Dada una lista de subconjuntos de atributos candidatos a super-reductos L_R, en la cual cada subconjunto tiene tamaño r, tal que $r <= K$, de la cual solo se han eliminado subconjuntos de atributos que no pertenecen a algún reducto (para considerar el menor número de super-reductos posibles). Para verificar si existen super-reductos de tamaño K basta con generar todos los subconjuntos de tamaño $K - r$ y verificar para cada subconjunto R' generado si existen candidatos a super-reductos $R \in L_R$, tal que, $R \cap R' = \emptyset$ y $R \cup R'$ es un super-reducto.

La idea presentada con anterioridad es una generalización de las estrategias de poda basadas en el core y CMA discutidas en este capítulo, ya que si el core no es vacío y tiene tamaño r entonces L_R solo contendrá al core y en el caso de que el core sea vacío, si consideramos $r = 1$ tenemos que: $L_R = \{\{c_q\} \mid c_q \in CMA \cap C'\}$.

Los subconjuntos de atributos en L_R se generarán a partir del conjunto de atributos C' como una secuencia ascendente acorde al orden en el cual se generaron los subconjuntos de la partición del conjunto de atributos de la MDBS según el Algoritmo 4.2. En la Definición 4.3 se define el orden que siguen los atributos en los candidatos generados, de esta forma se asegura que no se generen subconjuntos de atributos repetidos.

Definición 4.3. *Dada una partición del conjunto de atributos* $L_S = \{S_1, S_2, ..., S_t, ..., S_N\}$ *como resultado de aplicar el Algoritmo 4.2 a una MDBS y* $C' = \{c_{q_1}, c_{q_2}, ..., c_{q_t}, ..., c_{q_N}\}$, *en donde* c_{q_i} *es el primer atributo en el subconjunto* S_i *de* L_S. *Un subconjunto de atributos* $R = \{c_{q_{f_1}}, c_{q_{f_2}}, c_{q_{f_3}}, ..., c_{q_{f_r}}\}$ *de tamaño* r *cumple con el orden requerido si se cumple que:* $f_1 < f_2 < ... < f_3 < ... < f_r$.

Para generar candidatos a super-reductos que cumplan la Definición 4.3 es necesario que las columnas pertenecientes al core o al CMA sean las primeras en el conjunto C', ya que cada candidato a super-reducto generado comenzará por el core o por alguna combinación de los atributos en el CMA. Para lograr dicho resultado las columnas en la MDBS serán ordenadas de la siguiente forma:
Si el core no es vacío, las columnas pertenecientes al mismo serán las primeras columnas en la MDBS, en caso de que el core sea vacío, las columnas pertenecientes al CMA serán las primeras en la MDBS, a esta nueva MDBS la denotaremos como M' . En la Tabla 4.4 se muestra la MDBS en la Tabla 4.2 ordenada utilizando el procedimiento anterior.

Al generar una partición L_S de los atributos siguiendo el Algoritmo 4.2 partiendo de la MDBS ordenada como se indicó anteriormente y conociendo que C' está formado a partir del primer atributo de cada uno de los subconjuntos en L_S, se tiene que: si existe el core y éste tiene tamaño t los primeros t atributos en C' pertenecerán al core, si el core no existe, los primeros p atributos pertenecerán al CMA, donde p es el número de atributos en el subconjunto de atributos $CMA \cap C'$.

c_5	c_1	c_2	c_3	c_4
0	0	1	0	1
0	1	0	1	0
1	0	0	0	0
0	0	1	1	0

Tabla 4.4: Matriz de Discernibilidad Binaria Simplificada ordenada asociada a la MDBS de la Tabla 4.2

Ejemplo 4.2. *Partiendo del resultado que se muestra en la Tabla 4.3, tenemos que la partición del conjunto de atributos C' es: $[\{c_3, c_1\}, \{c_2, c_4\}, \{c_5\}]$, por tanto el conjunto*

de atributos C' es, $\{c_3, c_2, c_5\}$. Luego, como existe el core, los primeros atributos en cada candidato a super-reducto son los atributos que están en el core, un posible candidato a super-reducto sería $\{c_5, c_3\}$, pero no se cumple la Definición 4.3. Por el contrario si aplicamos el Algoritmo 4.2 a la MDBS de la Tabla 4.4, se obtiene la partición $L_S = [\{c_5\}, \{c_3, c_1\}, \{c_2, c_4\}]$, en donde el conjunto C' es: $\{c_5, c_3, c_2\}$ y $\{c_5, c_3\}$ es un candidato a super-reducto, ya que se puede verificar que la Definición 4.3 se cumple.

La estrategia que se sigue para generar la lista de candidatos consiste en ir almacenando subconjuntos de atributos partiendo de los subconjuntos de menor tamaño posible, luego iterativamente se remplaza cada subconjunto de atributos almacenado por subconjuntos de mayor tamaño, los nuevos subconjuntos de atributos se forman agregando un nuevo atributo al subconjunto remplazado.
En un inicio el primer candidato a super-reducto almacenado será el core en caso de no ser vacío y no ser un super-reducto (si el core es un super-reducto el procedimiento termina retornando el core como reducto más corto y una lista de candidatos vacía), si el core es vacío, cada atributo en el $CMA \cap C'$ se adicionará como un subconjunto candidato a super-reducto luego de verificar que no se cumple la condición de super-reducto (en caso de ser un super-reducto el procedimiento termina retornando un reducto más corto y una lista de candidatos vacía). Luego, iterativamente se elimina el primer subconjunto de atributos R y se adicionan al final los subconjuntos candidatos a super-reductos formados al agregar uno de los atributos en C' a R de forma tal que se cumpla la Definición 4.3 y la Proposición 2.1. Este procedimiento continúa hasta encontrar un super-reducto o se llegue a un número de candidatos predeterminado (siempre y cuando se hayan generado todos los candidatos a super-reductos del tamaño actual).

En el Algoritmo 4.3 se muestra el pseudocódigo del algoritmo de generación de *candidatos a super-reductos*. En el cual el procedimiento *índice_próximo_atributo*, recibe

como parámetro un candidato a super-reducto R y retorna, de existir, el índice i en C' del primer atributo c_{q_i} a adicionar a R para formar el nuevo candidato $R \cup c_{q_i}$ a verificar, el índice i hace referencia al atributo que le sigue en orden en C' al último atributo en R. Si en el proceso de generación de subconjuntos candidatos a super-reductos se encuentra un super-reducto, éste es un reducto más corto, por lo tanto se eliminan de la lista los subconjuntos de atributos del mismo tamaño del super-reducto hallado, ya que estos no son reductos más cortos, quedando en la lista solo posibles subconjuntos que pueden ser reductos más cortos. Un aspecto a tener en consideración en el algoritmo propuesto para calcular los candidatos a super-reductos, es que en éste, se generan los candidatos en base a un número de candidatos definido con anterioridad, por lo tanto, para asegurar que se generen todos los candidatos de un tamaño determinado el algoritmo solo finaliza si el número de candidatos en la lista de candidatos es mayor al número de candidatos predefinidos, y el primero y el último subconjunto de atributos en la lista de subconjuntos tienen igual tamaño, las funciones *t_primero* y *t_último* calculan el tamaño del primero y del último subconjunto de atributos respectivamente. En la sección de experimentos se muestra como se selecciona del número de candidatos a generar.

A continuación en el Ejemplo 2 se muestra el resultado de aplicar el Algoritmo 4.3 a la matriz de discernibilidad binaria simplificada de la Tabla 4.2.

Ejemplo 4.3. *Dada la MDBS de la Tabla 4.2, a la cual denominaremos M y conociendo que:*

- $CORE(M) = \{c_5\}$.
- $L_S = [\{c_5\}, \{c_3, c_1\}, \{c_2, c_4\}]$, *es la lista de subconjuntos de atributos generados por el Algoritmo 4.2 sobre M′ (M ordenada), ver Tabla 4.4, y* $C' = \{c_5, c_3, c_2\}$.

Algoritmo 4.3: Generación de candidatos a super-reductos

Entrada: Una $MDBS$ M, la lista de subconjuntos de atributos L_S generados según el Algoritmo 4.2 a partir de M' (M ordenada), cantidad máxima de candidatos a generar NC, el core de M, el CMA de M

Salida : Una lista de candidatos a super-reductos L_R, El primer reducto encontrado R

```
1  L_R ← {}
2  R ← {}, no se ha encontrado ningún reducto
3  C' ← { primer atributo en cada S_i ∈ L_S}
4  N, número de subconjuntos en L_S
5  si core ≠ ∅ entonces
6  |  si core es un super-reducto entonces
7  |  |  devolver L_R, core
8  |  fin
9  |  en otro caso
10 |  |  L_R ← core
11 |  fin
12 fin
13 en otro caso
14 |  para i ← 0 a número de atributos en el CMA ∩ C' hacer
15 |  |  c_q ← C'[i]
16 |  |  si {c_q} es un super-reducto entonces
17 |  |  |  R ← {c_q}
18 |  |  |  L_R ← {}
19 |  |  |  devolver L_R, R
20 |  |  fin
21 |  |  adicionar {c_q} a L_R
22 |  fin
23 fin
24 do
25 |  cand_r ←primer candidato de L_R
26 |  eliminar cand_r de L_R
27 |  índice_atributo ← índice_próximo_atributo(C',cand_r)
28 |  para i ← índice_atributo a N hacer
29 |  |  c_q ← C'[i]
30 |  |  si cand_r ∪ {c_q} es un super-reducto entonces
31 |  |  |  R ← cand_r ∪ {c_q}
32 |  |  |  eliminar de L_R los subconjuntos del mismo tamaño que R
33 |  |  |  devolver L_R, R
34 |  |  fin
35 |  |  en otro caso
36 |  |  |  si contribuye({c_q},cand_r) entonces
37 |  |  |  |  nuevo_candidato ← cand_r ∪ {c_q}
38 |  |  |  |  adicionar nuevo_candidato a L_R
39 |  |  |  fin
40 |  |  fin
41 |  fin
42 while número de candidatos en L_R < NC y t_primero(L_R) ≠ t_último(L_R)
43 devolver L_R, R
```

- $L_R = \{\}$, *es la lista inicial de candidatos a super-reductos generados.*
- $R = \emptyset$, *inicialmente no se ha encontrado ningún super-reducto.*

Como el core no es vacío, el primer candidato generado es $\{c_5\}$, *pero no es un super-reducto, por tanto,* $L_R = \{\{c_5\}\}$. *Luego se extrae el primer subconjunto de atributos candidato a super-reducto* $R = \{c_5\}$, *a continuación para cada atributo* $c_{q_i} \in C'$ *que le sigue en orden al último atributo en* R *se verifica si* $R \cup c_{q_i}$ *es un super-reducto o si* c_{q_i} *contribuye a* R, *es decir* c_3, c_2 *son los atributos que le siguen en orden a* c_5 *en* C', *por tanto se verifica según la Definición 2.10 que tanto* c_3 *como* c_2 *contribuyen a* $\{c_5\}$ *pero ninguno forma un super-reducto, ver el Tabla 4.2. A continuación los candidatos verificados son adicionados a* $L_R = \{\{c_5, c_3\}\{c_5, c_2\}\}$.
En la próxima iteración se extrae el subconjunto de atributos $R = \{c_5, c_3\}$ *y se puede observar que* c_2 *es el único atributo que le sigue a* c_3 *en* C', *el cual contribuye a* $\{c_5, c_3\}$ *y que además es un super-reducto. Por tanto, el procedimiento termina retornando* $L_R = \{\{c_5, c_2\}\}$ *y el reducto más corto* $R = \{c_5, c_3, c_2\}$.
Nota: Como es un ejemplo muy sencillo se llega a encontrar un reducto más corto en el proceso de generación de la lista de candidatos a super-reductos. En la mayoría de los casos esto no sucede, por lo tanto se retornaría la lista de candidatos L_R *y* $R = \emptyset$.

Discutidas las estrategias de poda que se utilizarán en esta tesis, estamos en condiciones de introducir el procedimiento que se utiliza para verificar por la existencia de super-reductos para un tamaño determinado.

4.2.4 Determinación de super-reductos de tamaño *K*

Los candidatos a super-reductos de tamaño K se generan a partir de la lista de candidatos que se obtiene al aplicar el Algoritmo 4.3 de la siguiente forma:

Partiendo de una lista de candidatos a super-reductos $L_R = \{R_1, R_2, ..., R_{NC}\}$ generada a partir del Algoritmo 4.3, en donde R_j es el candidato a super-reducto *j-ésimo* y dada una lista de subconjuntos de atributos $L_S = [S_1, S_2, ..., S_N]$ generada a partir del Algoritmo 4.2, donde S_i es el subconjunto de atributos *i-ésimo* en el cual la columna correspondiente al primer atributo es supercolumna de cada una de las columnas correspondientes al resto de los atributos en S_i y $C' = \{c_{q_i} | c_{q_i}$ *es el primer atributo en* $S_i\}$.

En un inicio se extrae el primer candidato a super-reducto de la lista de candidatos $R_1 = \{c_{q_{f_1}}, c_{q_{f_2}}, ..., c_{q_{fr}}\}$, donde $f_h = 1, 2, .., N$ y $h = 1, 2, ...r$. Luego, se van generando todos los subconjuntos de atributos R' de tamaño $K - r$ considerando solo los atributos $c_{q_i} \in C'$ para los cuales $i > f_r$, luego para cada subconjunto R' se crea un subconjunto de tamaño K en el cual los primeros r atributos pertenecen a R_1 y los restantes a R', es decir, los subconjuntos candidatos a super-reductos de tamaño K serían de la forma $R_1 \cup R'$. Los candidatos a super-reductos se verifican a medida que se van generando, por lo que si en algún punto de la generación de candidatos se encuentra un super-reducto el proceso termina. Si al generar todos los candidatos a super-reductos a partir de R_1, no se encuentra algún super-reducto, se procede a formar nuevos candidatos a partir de los restantes candidatos en L_R, este proceso continúa hasta que se encuentre un super-reducto de tamaño K, o se utilicen todos los candidatos en L_R para los cuales se cumple que $f_r + (K - r)$ es menor al número de atributos en C'. Si no se encuentran super-reductos de tamaño K al utilizar los candidatos en L_R significa que no existen super-reductos de tamaño K.

Los subconjuntos de atributos R' de tamaño $t = K - r$ se generan de acuerdo al orden de los atributos en C' de la siguiente forma:
El último subconjunto posible a generar estará formado por los últimos t atributos en

C', $\{c_{q_{l_1}}, c_{q_{l_2}}, ..., c_{q_{l_t}}\}$. Y el primer subconjunto (subconjunto actual) por los primeros t atributos en C', $\{c_{q_{g_1}}, c_{q_{g_2}}, ..., c_{q_{g_t}}\}$. Los subconjuntos se van generando desde el primero al último como sigue:

Para generar el próximo subconjunto se busca el atributo $c_{q_{g_i}}$ más a la derecha del subconjunto actual $\{c_{q_{g_1}}, c_{q_{g_2}}, ..., c_{g_{g_t}}\}$, que cumpla que existe $l_i > g_i$, para $0 < i \leq t$. Entonces g_i se incrementa en uno y para cada $g_j > g_i$, $c_{q_{g_j}} = c_{q_s}$, tal que $c_{q_s} \in C'$ y $s = (q_j - 1)$. Si $c_{q_{g_i}}$ no existe es que el próximo subconjunto es el último.

De forma general el primer subconjunto de tamaño K generado siguiendo el procedimiento anterior será $R_1 \cup \{c_{q_i}, c_{q_{(i+1)}}, ..., c_{q_{(i+t)}}\}$, donde $i = f_r + 1$ y $i + t < N$. El último subconjunto de atributos a generar será $R_j \cup R'$, en donde R_j es el subconjunto más a la derecha en L_R para el cual c_{q_l} es el último atributo en R_j y se cumple que $l + t < N$ y el subconjunto de atributos $R' = \{c_{q_{N-t}}, c_{q_{N-t-1}}, .., c_{q_N}\}$.

4.2.5 Algoritmo para determinar el tamaño de los reductos más cortos

A continuación en el Algoritmo 4.4 se muestra el pseudocódigo del algoritmo que se utiliza para calcular el tamaño de los reductos más cortos partiendo de una estimación inicial de su tamaño y utilizando las estrategias de poda que se abordaron la Sección 4.2.3. En el algoritmo la primera condición a verificar es si en el proceso de generación de candidatos a super-reductos se encontró un super-reducto, ya que si es el caso, el procedimiento termina retornando el super-reducto y su respectivo tamaño, ya que este será un reducto más corto. Luego, se fija como cota inferior del tamaño de los reductos más cortos el tamaño de los candidatos en L_R, en donde la función $tamaño\_candidatos(L_R)$ retorna el tamaño de los candidatos a super-reductos en L_R.

Algoritmo 4.4: Cómputo del tamaño de los reductos más cortos

Entrada: Una MDBS M, la lista de subconjuntos de atributos L_S generados según el Algoritmo 4.2 a partir de M' (M ordenada), primer reducto más corto encontrado R, lista de candidatos a reductos L_R, estimación inicial del tamaño de los reductos más corto K

Salida : tamaño de los reductos más cortos k, primer reducto más corto encontrado R

```
si R ≠ ∅ entonces
    k ← tamaño de R
    devolver k, R
fin
si K ≤ tamaño_candidatos(L_R) entonces
    K ← tamaño_candidatos(L_R) + 1
fin
reducto_temporal ← buscar_super_reducto(M,L_S,L_R,∅,K)
si reducto_temporal ≠ ∅ entonces
    mientras reducto_temporal ≠ ∅ hacer
        R ← reducto_temporal
        K ← K − 1
        reducto_temporal ← buscar_super_reducto(M,L_S,L_R,∅,K)
    fin
    k ← K + 1
fin
en otro caso
    mientras reducto_temporal = ∅ hacer
        K ← K + 1
        reducto_temporal ← buscar_super_reducto(M,L_S,L_R,∅,K)
    fin
    k ← K
    R ← reducto_temporal
fin
devolver k, R
```

La función *buscar_super_reducto* en el algoritmo retorna el primer super-reducto (según el orden en el que se generan al seguir la estrategia abordada con anterioridad) de tamaño K encontrado, utilizando la lista de candidatos a reductos y la lista de subconjuntos de atributos para verificar el menor número de candidatos posibles, adicionalmente la función puede recibir como parámetro un subconjunto de atributos de tamaño K en el cual comenzará la búsqueda, en caso de ser el subconjunto vacío la función generará todos los subconjuntos de tamaño K desde el primer subconjunto de atributos hasta encontrar un super-reducto o alcanzar el último subconjunto de tamaño K, ver Sección 4.2.4. Al aplicar el algoritmo obtenemos el tamaño de los reductos más

cortos y el primer reducto más corto encontrado.

4.3 Cálculo de todos los reductos más cortos

La idea central del algoritmo que proponemos para el cálculo de todos los reductos más cortos es que si conocemos el tamaño de los reductos más cortos solo se deben verificar las combinaciones de dicho tamaño. Como conocemos el tamaño de los reductos más cortos, si un subconjunto de dicho tamaño es un super-reducto éste va a ser un reducto más corto. Con esto en mente, el procedimiento que se utiliza para realizar el cálculo de todos los reductos más cortos es similar al utilizado para verificar por la existencia de super-reductos de un tamaño dado, ver Figura 4.3, con la diferencia de que, al encontrar un super-reducto, el procedimiento tiene que continuar hasta verificar por todos los posibles candidatos a super-reductos de dicho tamaño. El algoritmo procede a calcular todos los super-reductos, dado su tamaño y el primer reducto más corto encontrado, siguiendo el mismo orden de generación de candidatos que se utilizó para determinar el tamaño de los reductos más cortos, ver la Sección 4.2. Otro aspecto importante es que al encontrar un super-reducto se tendrá que verificar si se pueden formar super-reductos con los restantes atributos de los subconjuntos a los cuales pertenecen los atributos en el super-reducto hallado, es decir:

Dada la partición del conjunto de atributos $L_S = [S_1, S_2, ..., S_N]$ y un super-reducto $R = \{c_{f_1}, c_{f_2}, ..., c_{f_r}\}$, tal que c_{f_l} es un atributo en S_{f_l}. Los candidatos generados a partir de R se generan de la siguiente forma:

Primero se verifica si para algún $c_{f_l} \in R$, S_{f_l} contiene más de un atributo. Dado que si cada atributo en R pertenece a un subconjunto en la partición L_S formado por un único atributo, R será el único super-reducto utilizando dichos subconjuntos.

En caso de que no suceda lo anterior, un nuevo candidato a super-reducto se genera a

partir de R como se explica a continuación:

Se selecciona el atributo c_{f_l} más a la derecha en R que no sea el último en su subconjunto y se remplaza por el atributo que le sigue en orden en S_{f_l} y cada $c_{f_j} \in R$ con $j > l$ será remplazado por el primer atributo en el subconjunto al cual pertenece. Los nuevos candidatos se generan a partir del anterior siguiendo el mismo procedimiento hasta que se genere un candidato formado por los últimos atributos de cada subconjunto en L_S, el cual será el último candidato posible a generar a partir de R.

Ejemplo 4.4. *Dado un super-reducto* $R = \{c_1, c_5, c_9\}$*, y dada la partición del conjunto de atributos representado por* $L_S = [\{c_1, c_2, c_3\}, \{c_5, c_6\}, \{c_9, c_7, c_8\}]$*, los candidatos a super-reductos generados a partir de* R *siguiendo el procedimiento anterior se muestran a continuación :*

$$\longmapsto \{c_1, c_5, c_7\} \longrightarrow \{c_1, c_5, c_8\} \longrightarrow \{c_1, c_6, c_9\} \longrightarrow \{c_1, c_6, c_7\} \longrightarrow \{c_1, c_6, c_8\} \curvearrowright$$
$$\longrightarrow \{c_2, c_5, c_9\} \longrightarrow \{c_2, c_5, c_7\} \longrightarrow \{c_2, c_5, c_8\} \longrightarrow \{c_2, c_6, c_9\} \longrightarrow \{c_2, c_6, c_7\} \curvearrowright$$
$$\longrightarrow \{c_2, c_6, c_8\} \longrightarrow \{c_3, c_5, c_9\} \longrightarrow \{c_3, c_5, c_7\} \longrightarrow \{c_3, c_5, c_8\} \longrightarrow \{c_3, c_6, c_9\} \curvearrowright$$
$$\longrightarrow \{c_3, c_6, c_7\} \longrightarrow \{c_3, c_6, c_8\}$$

A continuación en el Algoritmo 4.5 se muestra el pseudocódigo del algoritmo para calcular todos los reductos más cortos, en donde *buscar_super_reducto* es la misma función utilizada para verificar por la existencia de super-reductos de tamaño K, en este caso se le indica que la búsqueda comience en el super-reducto R, por lo tanto retornará el super-reducto posterior a R. En el algoritmo la función *siguiente_sr_subconjuntos*, dado un super-reducto R, retorna el siguiente super-reducto considerando los restantes elementos de los subconjuntos en L_S a los cuales pertenecen los atributos en R.

Algoritmo 4.5: Búsqueda de todos los reductos más cortos

Entrada: Una MDBS M, la lista de subconjuntos de atributos L_S generados según el Algoritmo 4.2 a partir de M' (M ordenada), lista de candidatos a reductos L_R, primer reducto más corto R, tamaño del reducto más corto k
Salida : Todos los reductos más cortos LR

```
1  reducto_temporal ← R
2  mientras reducto_temporal ≠ ∅ hacer
3      adicionar reducto_temporal a LR
4      reducto_temporal ← siguiente_sr_subconjuntos(M,L_S,reducto_temporal)
5      mientras reducto_temporal ≠ ∅ hacer
6          adicionar reducto_temporal a LR
7          reducto_temporal ← siguiente_sr_subconjuntos(M,L_S,reducto_temporal)
8      fin
9      reducto_temporal ← buscar_super_reducto(M,L_S,L_R,R,k)
10     R ← reducto_temporal
11 fin
12 devolver LR
```

A continuación se ejemplifica el procedimiento general a seguir para calcular todos los reductos más cortos en una MDBS.

Ejemplo 4.5. *Dada la MDBS de la Tabla 4.2, conociendo el primer reducto más corto* $R = \{c_5, c_3, c_2\}$ *de tamaño 3, la lista de candidatos a reductos* $L_R = \{\{c_5, c_2\}\}$ *y la lista de subconjuntos de atributos* $L_S = [\{c_5\}, \{c_3, c_1\}, \{c_2, c_4\}]$*, ver Ejemplo 4.2, se procede como sigue:*

Primero se verifica la condición de super-reducto para cada posible candidato generado a partir de los restantes atributos de cada subconjunto en L_S *a los cuales pertenecen los atributos en* R *(de la línea 4 a la línea 8 del Algoritmo 4.5). El primer candidato a verificar es* $\{c_5, c_1, c_2\}$*, que en efecto es un super-reducto, luego se verifica* $\{c_5, c_1, c_4\}$*, que no es un super-reducto, a continuación se verifica* $\{c_5, c_3, c_4\}$ *que sí es un super-reducto. Al no existir más candidatos por generar a partir de* R *se procede a buscar el próximo super-reducto de tamaño* 3 *que le sigue en orden a* R*, como el tamaño de los candidatos en* L_R *es de 2, tenemos que* $R = \{c_5, c_3\} \cup \{c_2\}$*, donde el subconjunto* $\{c_2\}$ *es el último subconjunto* R' *a generar, por tanto utilizando el subconjunto* $\{c_5, c_3\}$ *no se pueden generar mas candidatos a super-reductos de tamaño 3. Luego, se procede a*

utilizar el próximo candidato en la lista de candidatos a super-reductos L_R. Se parte del candidato $\{c_5, c_2\}$ y se observa que el subconjunto al cual pertenece el atributo c_2 es el último subconjunto en L_S, por lo tanto no es posible formar candidatos a super-reductos de tamaño 3 a partir de $\{c_5, c_2\}$, por lo que no restan candidatos a super-reductos por verificar. Al finalizar los reductos más cortos son: $\{c_5, c_3, c_2\}$, $\{c_5, c_1, c_2\}$ y $\{c_5, c_3, c_4\}$.

4.4 Análisis de Complejidad

Para realizar el análisis de complejidad del algoritmo propuesto se considera el peor caso al generar la partición del conjunto de atributos utilizando el Algoritmo 4.3, esto sucede si cada uno de los subconjuntos en la partición generada está formado por un único atributo, entonces $N = n$, donde N es el número de subconjuntos de atributos resultantes de realizar una partición del conjunto de atributos utilizando el Algoritmo 4.3 y n es el número de atributos en la Matriz de Discernibilidad Binaria Simplificada *(MDBS)*.

El término predominante en la ecuación que expresa la complejidad temporal del algoritmo propuesto corresponde al proceso de generación y procesamiento de los subconjuntos de atributos. Por lo tanto, se inicia este análisis calculando el número de subconjuntos de atributos procesados por el algoritmo. El algoritmo procesa los subconjuntos de atributos en orden creciente o decreciente dependiendo de la estimación del tamaño de los reductos más cortos. Sea k la longitud del primer reducto más corto encontrado y K la estimación del tamaño de los reductos más cortos, a continuación se muestra el número de subconjuntos procesados en cada etapa del algoritmo.

Primero, el algoritmo genera una lista de candidatos a super-reductos que se utiliza para podar el espacio de búsqueda. Supongamos que la lista de candidatos contiene

subconjuntos de tamaño r, como los subconjuntos en la lista de candidatos se generan en orden creciente, el proceso de generación de candidatos terminará al generar todas las combinaciones de atributos de longitud r. La Ecuación 4.6 expresa el máximo número de subconjuntos de atributos procesados al generar la lista de candidatos como la suma de los primeros r coeficientes binomiales para un n determinado.

$$NS_{LR}(n,r) = \sum_{i=1}^{r} C_i^n \tag{4.6}$$

Luego, a partir de la lista de candidatos y de la estimación inicial del tamaño de los reductos más cortos se procede a calcular el tamaño real de los reductos más cortos y posteriormente, conociendo el tamaño real de los reductos más cortos el algoritmo calcula todos los super-reductos de dicho tamaño. La Ecuación 4.7 expresa el máximo número de subconjuntos de atributos procesados antes de encontrar todos reductos más cortos en el peor caso, que se da si la estimación K del tamaño de los reductos más cortos es diferente al tamaño real k de los reductos más cortos y la longitud r de los subconjuntos en la lista de candidatos generada con el Algoritmo 4.3 es menor que $k-1$.

$$NS_{CR}(n,r,k,K) = C_r^n * \begin{cases} \sum_{i=k-1}^{K} C_{i-r}^n \ , \ si\ k < K \\ \sum_{i=K}^{k} C_{i-r}^n \ , \ si\ k > K \end{cases} \tag{4.7}$$

La primera parte de la Ecuación 4.7 expresa el máximo número de subconjuntos de atributos procesados siguiendo la estrategia decremental y la segunda parte expresa el máximo número de subconjuntos de atributos procesados siguiendo la estrategia incremental. En la Ecuación 4.7, C_r^n expresa el número máximo de candidatos a super-reductos que puede contener la lista de candidatos generada con el Algoritmo 4.3, ya que cada candidato a super-reducto a procesar se formará a partir de los candidatos a super-reductos en dicha lista de candidatos, ver Sección 4.2.4.

El algoritmo genera cada uno de los subconjuntos en la Ecuación 4.7 y evalúa la condición de super-reducto para cada subconjunto generado. Para hacer esto, se utiliza una función para generar el siguiente subconjunto de atributos en función del subconjunto de atributos actual. La función que verifica la condición de super-reducto tiene una complejidad temporal de $O(f*k)$, donde f es el número de filas en la MDBS, ya que busca una fila de ceros en la MDBS considerando solo los k atributos que pertenecen al subconjunto de atributos a evaluar. La función utilizada para generar el siguiente subconjunto de atributos tiene una complejidad temporal de $O(k-r)$ ya que modifica solo los últimos $k-r$ atributos del subconjunto actual. Por otra parte, para los subconjuntos de atributos dados por la Ecuación 4.6 se evalúa si el subconjunto de atributos luego de agregarle otro atributo es un super-reducto o puede ser un candidato a super-reducto, dicha verificación tiene una complejidad temporal de $O(f*r)$, pues debe verificar por filas los r atributos en las f filas de la MDBS.

La Ecuación 4.8 expresa la complejidad temporal en el peor caso del Algoritmo 4.3, el cual genera la lista de candidatos a super-reductos. Mientras que la Ecuación 4.9 expresa la complejidad temporal en el peor caso de la búsqueda del tamaño real de los reductos más cortos y del cálculo de todos los reductos más cortos dado su tamaño. Al final, la Ecuación 4.10 expresa la complejidad temporal en el peor caso del algoritmo propuesto mediante la suma del tiempo de creación de la lista de candidatos a super-reductos Ecuación 4.8 y la Ecuación 4.9 que expresa el tiempo de búsqueda del tamaño de los reductos más cortos y el cálculo de todos los reductos más cortos dado su tamaño.

$$T_{LR}(n,r,k) = (O(f*r) * NS_{LR}(n,r)) \tag{4.8}$$

$$\begin{aligned} T_{CR}(n,r,k,K) =& (O(f*k)+O(k-r))(NS_{CR}(n,r,k,K)) \\ &= O((f*r)*NS_{LR}(n,r)+(k-r)*NS_{CR}(n,r,k,K)) \end{aligned} \tag{4.9}$$

$$T_{AP}(n,r,k,K) = T_{LR}(n,r,k)+T_{CR}(n,r,k,K) \tag{4.10}$$

4.5 Síntesis

En este capítulo se describió detalladamente cada uno de los pasos que conforman el algoritmo propuesto para calcular todos los reductos más cortos. La idea central del algoritmo propuesto es que si conocemos el tamaño de los reductos más cortos solo se deben verificar las combinaciones de dicho tamaño. En la primera sección de este capítulo se describió el mecanismo utilizado para obtener una aproximación del tamaño de los reductos. Mientras que en las secciones segunda y tercera se describieron a detalle los mecanismos de poda utilizados en esta tesis y cada una de las partes del algoritmo utilizado para hallar todos los reductos más cortos de un sistema de decisión. Finalmente se presentó el análisis de complejidad del algoritmo propuesto.

Capítulo 5

Experimentos

En este capítulo se describen los experimentos realizados para evaluar el algoritmo que se introduce en esta tesis para el cálculo de todos los reductos más cortos. Primeramente, en la sección Sección 5.1 se detalla los conjuntos de datos utilizados para evaluar el algoritmo propuesto. Luego, en la Sección 5.2 se muestran los experimentos realizados con el objetivo de mostrar el desempeño del algoritmo propuesto en está tesis, incluyendo una comparación con el estado del arte.

5.1 Descripción de los conjuntos de datos

Para evaluar, y realizar las comparaciones con el estado del arte se utilizaron 19 conjuntos de datos pertenecientes al repositorio de aprendizaje automático de la *UCI* [Dua and Graff, 2019]. Algunos de estos conjuntos de datos han sido utilizados por diferentes autores [Zhou et al., 2009; Piza-Dávila et al., 2020] para evaluar la eficiencia de los algoritmos propuestos. Para los atributos numéricos, se utilizó el método de discretización en bins de igual tamaño *(EWB)* de Weka, como se describe en [Rajalakshmi et al., 2016]. En la Tabla 5.1 se muestra una descripción de los conjuntos de datos utilizados para realizar los experimentos, en la cual, las columnas: *Objetos*, *Atributos* y *Clases* muestran el número de objetos, el número de atributos y el número de clases que conforman cada conjunto de datos respectivamente y *Filas, Columnas y Densidad* muestran el número de filas, número de columnas y la densidad de la matriz de discernibilidad binaria simplificada asociada a cada conjunto de datos. El algoritmo propuesto trabaja sobre la Matriz de Discernibilidad Binaria Simplificada *(MDBS)*, la

cual contiene solo las filas básicas de la matriz de discernibilidad binaria, que se obtiene de la comparación entre los atributos para cada par de objetos que pertenecen a clases diferentes en el conjunto de entrenamiento, ver Sección 2.6. También se han incluido 6 matrices de discernibilidad binaria simplificada sintéticas en los experimentos (*'1500x120'*,*'197x45'*, *'191x53'*, *'197x53'*, *'178x54'* y *'200x53'*), con el fin de explorar el comportamiento de los algoritmos con matrices más complejas.

Tabla 5.1: Descripción de los conjuntos de datos utilizados.

Conjunto de Datos	Objetos	Atributos	Clases	Filas	Columnas	Densidad
audiology	226	69	24	302	69	0.0692
biodeg	1055	41	2	40	41	0.0835
chess	3196	36	2	30	36	0.0361
connect	6756	42	3	406	42	0.0476
dermatology	366	34	6	1103	34	0.3445
flags	194	29	8	390	29	0.3536
kr-vs-kp	3196	36	2	29	36	0.0297
landsat	6435	36	7	9815	36	0.3318
lung-cancer	56	32	3	237	56	0.4589
molecular_Promoter	106	58	2	2761	57	0.7516
mushroom	8124	22	2	39	22	0.3019
postoperative-patient	90	8	3	8	8	0.1250
soybean-l	307	35	19	49	35	0.1207
soybean-s	47	35	4	99	35	0.2733
spect	267	22	2	17	22	0.0535
sponge	76	45	3	68	45	0.3824
student-mat	395	32	20	6904	32	0.4349
student-por	294	32	20	8158	32	0.4138
trains	10	32	2	21	32	0.3289
1500x120	-	120	-	1500	120	0.7461
197x45	-	45	-	197	45	0.2012
191x53	-	53	-	191	53	0.2035
197x53	-	53	-	194	53	0.1989
178x54	-	54	-	178	54	0.1931
200x53	-	53	-	200	53	0.2020

5.2 Evaluación del algoritmo propuesto

Los experimentos fueron realizados en una computadora con procesador Intel Xeon a 2.40 GHz, sistema operativo Windows 10 de 64 bits, memoria RAM de 256 GB y la

programación del algoritmo propuesto en esta tesis se realizó en el lenguaje de programación *Java*, dado que el algoritmo contra el que se compara [Piza-Dávila et al., 2020] está implementado por sus autores en dicho lenguaje.

Antes de realizar la evaluación del algoritmo que se propone en esta tesis para realizar el cálculo de los reductos más cortos, dado que el algoritmo estima el tamaño de los reductos más cortos en la Sección 5.2.1 se realiza un estudio experimental del método utilizado para la estimación del tamaño de los reductos más cortos. Luego, en la Sección 5.2.2 se muestra como se realiza la selección del número de candidatos a super-reductos a generar. Después, en la Sección 5.2.3 se realiza una comparación del algoritmo propuesto con el algoritmo más recientemente reportado en la literatura [Piza-Dávila et al., 2020].

5.2.1 Estimación del tamaño de los reductos más cortos

Los primeros experimentos que se realizaron estuvieron enfocados evaluar el método de estimación del tamaño de los reductos más cortos y encontrar una configuración adecuada del mismo. Para esto se utilizó un ***modelo de regresión polinomial***, ver Sección 4.2, ya que éste es muy utilizado para predecir los valores de una variable cuantitativa a partir de los valores de otras variables también cuantitativas [Granados, 2016]. Para entrenar este modelo se se consideraron las 7 características extraídas de la matriz de discernibilidad binaria simplificada mencionadas en la sección Sección 4.1 del Capítulo 4. Y se generó un conjunto de datos sintéticos, tratando de cubrir el mayor número de posibles combinaciones de propiedades de las matrices de discernibilidad binaria simplificada, el conjunto de datos generado consta de 3000 instancias.

Para seleccionar un subconjunto de características del conjunto inicial se conside-

raron todos los subconjuntos posibles (127 subconjuntos), para cada subconjunto se evaluó el modelo utilizando validación cruzada de 10 pliegues y se seleccionó el subconjunto de características que obtuvo el menor error cuadrático medio *(MSE)*. Las características seleccionadas luego de realizar dicho procedimiento fueron: N_C, $DENS$, MED_F y $STDV_F$. A continuación, en la Tabla 5.2 se muestran los valores de estas características para las MDBS asociadas a los conjuntos de datos de la Tabla 5.1, en la cual, T_{MRC} y N_{RMC} son el tamaño y el número de reductos más cortos respectivamente y EST es el valor estimado del tamaño de los reductos más cortos.

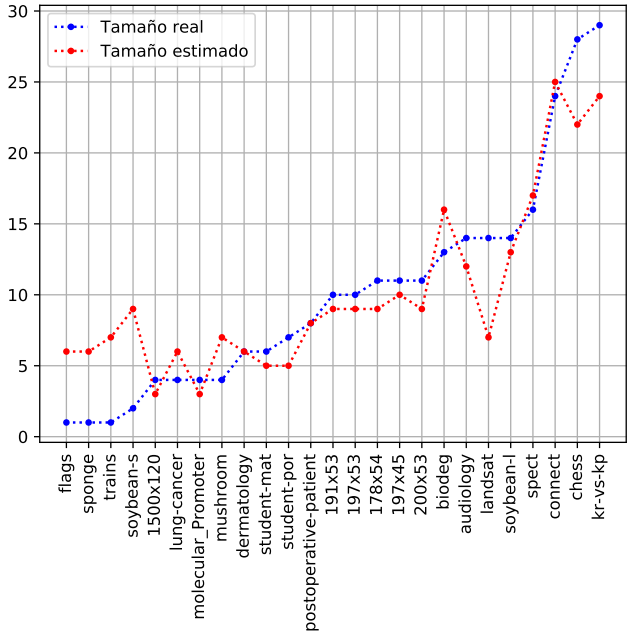
Tabla 5.2: Estimación del tamaño de los reductos más cortos realizada por el mejor modelo de regresión lineal obtenido.

MDBS	N_C	$DENS$	MED_F	$STDV_F$	T_{RMC}	EST	N_{RMC}
audiology	69	0.0692	5.0	1.8117	14	12	20
biodeg	41	0.0835	3.0	1.8150	13	16	2
chess	36	0.0361	1.0	0.6403	28	22	20
connect	42	0.0476	2.0	0.0000	24	25	10
dermatology	34	0.3445	12.0	3.5818	6	6	137
flags	29	0.3536	10.0	2.3204	1	6	1
kr-vs-kp	36	0.0297	1.0	0.2534	29	24	4
landsat	36	0.3318	12.0	2.4486	14	7	6
lung-cancer	56	0.4589	26.0	5.6822	4	6	112
molecular_Promoter	57	0.7516	43.0	3.1342	4	3	81
mushroom	22	0.3019	6.0	2.3695	4	7	3
postoperative-patient	8	0.1250	1.0	0.0000	8	8	1
soybean-l	35	0.1207	4.0	2.3842	14	13	1
soybean-s	35	0.2733	10.0	1.8377	2	9	4
spect	22	0.0535	1.0	0.3812	16	17	2
sponge	45	0.3824	16.0	5.9793	1	6	1
student-mat	32	0.4349	14.0	2.1465	6	5	2
student-por	32	0.4138	13.0	2.0381	7	5	1
trains	32	0.3289	11.0	2.2598	1	7	1
1500x120	120	0.7461	90.0	4.7960	4	3	37484
197x45	45	0.2012	9.0	2.6509	11	10	25
191x53	53	0.2035	11.0	2.7988	10	9	9
197x53	53	0.1989	11.0	2.9499	10	9	1
178x54	54	0.1931	10.0	2.7515	11	9	235
200x53	53	0.2020	11.0	2.8597	11	9	1056

El MSE es el criterio de evaluación más usado para problemas de regresión [Granados, 2016]. Sin embargo, el error cuadrático medio no es del todo intuitivo porque

nos da el error medio al cuadrado. Así que si nuestro modelo para estimar el tamaño de los reductos más cortos tiene un error cuadrático medio de 16 daría la impresión de que tiene mucho error. En realidad, el error sería ±4 en media, porque la raíz cuadrada de 16 es 4. Por tanto, para tener una idea mas intuitiva del error del modelo se utiliza la Raíz Cuadrada del Error Cuadrático Medio *(RMSE)*. Para evaluar el modelo de regresión lineal utilizado se utilizaron los conjuntos de datos en la Tabla 5.1, para los cuales el RMSE fue de 3.37, éste valor indica que el error de estimación del tamaño de los reductos más cortos en promedio para los conjuntos de datos en la Tabla 5.1 es de aproximadamente 3.

Figura 5.1: Estimación del tamaño de los reductos más cortos realizada por el mejor modelo de regresión lineal obtenido.



Para visualizar la diferencia entre el valor real del tamaño de los reductos más cortos y el tamaño estimado por el modelo de regresión lineal utilizado, se graficó el valor de la estimación del tamaño de los reductos más cortos y el tamaño real de dichos reductos para los conjuntos de datos de la Tabla 5.1. Como se puede apreciar en la Figura 5.1, en

la mayoría de los casos la estimación del tamaño de los reductos más cortos realizada por el modelo de regresión lineal está relativamente cercana al valor real del tamaño de los reductos más cortos.

5.2.2 Selección del número de candidatos a generar

El algoritmo propuesto tiene como parámetro el número de subconjuntos candidatos a utilizar en el proceso de búsqueda, ver Algoritmo 4.3. Este número de candidatos va a depender del número de subconjuntos en la partición del conjunto de atributos, por lo que si se fija un número excesivo de candidatos a generar, puede que el proceso de generación de candidatos se vuelva computacionalmente costoso y, por el contrario, si se considera un número demasiado pequeño, este conjunto no contribuirá al proceso de poda. Si bien el número de atributos en el conjunto de datos no es el único un factor que determina el número de candidatos a generar, ya que existen otras características de la MDBS que pueden contribuir a su determinación, en esta tesis se considera solo el número de atributos del conjunto de datos para determinar el número de candidatos a generar. Un estudio más completo considerando más características de la MDBS está fuera del ámbito de esta tesis. Por lo tanto, en el método propuesto, se decidió utilizar un número de candidatos basado combinaciones de N en k, C_k^N, dónde N es el número de subconjuntos en la partición del conjunto de atributos, ver Proposición 4.2. Por lo tanto, se consideraron como números de candidatos a generar C_3^N, C_4^N y C_5^N.

Este experimento consistió en probar el algoritmo propuesto variando el número de candidatos a generar. Para este experimento, se consideraron 3 números de candidatos distintos (C_3^N, C_4^N y C_5^N). La Tabla 5.3 muestra los tiempos de ejecución en segundos del algoritmo propuesto para algunos conjuntos de datos del repositorio UCI, considerando cada uno de los números de candidatos anteriores. Al final de este experimento, se

aplicó la prueba de Wilcoxon a las medias obtenidas para determinar si existe diferencia estadística, dando como resultado que no se rechaza la hipótesis nula, por lo que no hay evidencia suficiente para determinar que un número de candidatos sea mejor que otro. No obstante, se obtuvieron mejores resultados al utilizar C_4^N como número de candidatos a generar, es así que para los experimentos posteriores se tomó C_4^N como el número de candidatos a generar. De este experimento se puede concluir que utilizar un número mayor de candidatos a reductos no mejora necesariamente el rendimiento del algoritmo y en algunos casos empeora dicho rendimiento.

Tabla 5.3: Tiempos de ejecución en segundos del algoritmo propuesto para algunos conjuntos de datos del repositorio UCI, considerando diferentes tamaños para el número de candidatos a generar.

Conjunto de Datos	C_3^N	C_4^N	C_5^N
chess	0.009	0.010	0.008
connect	0.698	0.680	1.233
dermatology	0.270	0.280	0.279
flags	0.006	0.005	0.006
kr-vs-kp	0.010	0.009	0.009
lung-cancer	0.098	0.098	0.099
molecular_Promoter	1.415	1.583	1.339
mushroom	0.008	0.007	0.007
postoperative-patient	0.002	0.002	0.002
soybean-l	0.031	0.021	0.021
soybean-s	0.011	0.008	0.007
spect	0.008	0.007	0.006
sponge	0.004	0.007	0.005
student-mat	0.568	0.535	0.554
student-por	1.794	1.645	2.468
trains	0.006	0.005	0.005
Promedio	0.308	0.306	0.378

5.2.3 Comparación con el estado del arte

Para ***evaluar la eficiencia*** del algoritmo propuesto se realizó una comparación contra el algoritmo más recientemente reportado en la literatura para el cálculo de todos

los reductos más cortos, presentado en [Piza-Dávila et al., 2020], comparando el tiempo de ejecución y los requerimientos de memoria, para ello se utilizaron los conjuntos de datos que se muestran en la Tabla 5.1.

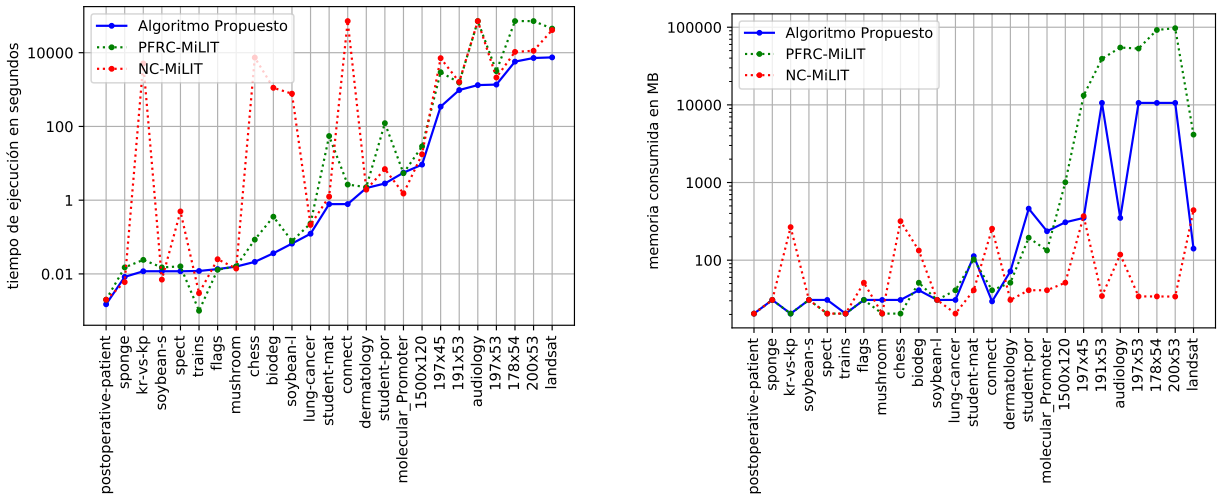
El algoritmo presentado en [Piza-Dávila et al., 2020] denominado *MLIT* utiliza dos estrategias para realizar la búsqueda de los reductos más cortos y decide cuál aplicar en función de la densidad de la matriz de discernibilidad binaria reducida. La primera estrategia denominada *PFRC-MiLIT* se aplica si la densidad de la matriz es menor 0.3, la otra estrategia denominada *NC-MiLIT* se aplica si la densidad de la matriz es mayor o igual que 0.3. Con el objetivo de realizar una comparación más extensa se consideró cada estrategia por separado.

En la Figura 5.2 a la izquierda se muestra el tiempo de ejecución en segundos de los algoritmos y a la derecha se muestra los requerimientos máximos de memoria en Megabytes de los mismos, al ser aplicados en los conjuntos de datos de la Tabla 5.1, en donde *PFRC-MiLIT* y *NC-MiLIT* son las estrategias introducidas en [Piza-Dávila et al., 2020]. Para el conjunto de datos *'audiology'* las estrategias presentadas en [Piza-Dávila et al., 2020] no son capaces de encontrar todos los reductos más cortos antes de *20 horas*, mientras que el algoritmo propuesto es capaz de encontrar todos los reductos más cortos para todos los conjuntos de datos en la Tabla 5.1. Por otra parte, para el conjunto de datos *'connect'* la estrategia *PFRC-MiLIT* encuentra solución relativamente rápido, al contrario de la estrategia *NC-MiLIT* que no es capaz de encontrar todos los reductos más cortos antes de *20 horas*, para los casos que no se encontró solución antes de *20 horas* se tomó en consideración el máximo de memoria consumida hasta ese momento. Como se puede observar, en la mayoría de los casos, el algoritmo propuesto en esta tesis calcula todos los reductos más cortos en un tiempo menor que las

estrategias propuestas en [Piza-Dávila et al., 2020], sin requerir demasiada memoria, especialmente en las matrices que requieren un mayor tiempo de ejecución.

Para medir el consumo máximo de memoria de los algoritmos se utilizó la clase *java.lang.Runtime* de *Java* [Mughal and Rasmussen, 2003], con excepción de los casos que no se encontró solución antes de 20 horas, para lo cual se utilizó *VisualVM* [Jiri Sedlacek, 2017] para monitorizar el consumo de memoria máximo hasta las 20 horas.

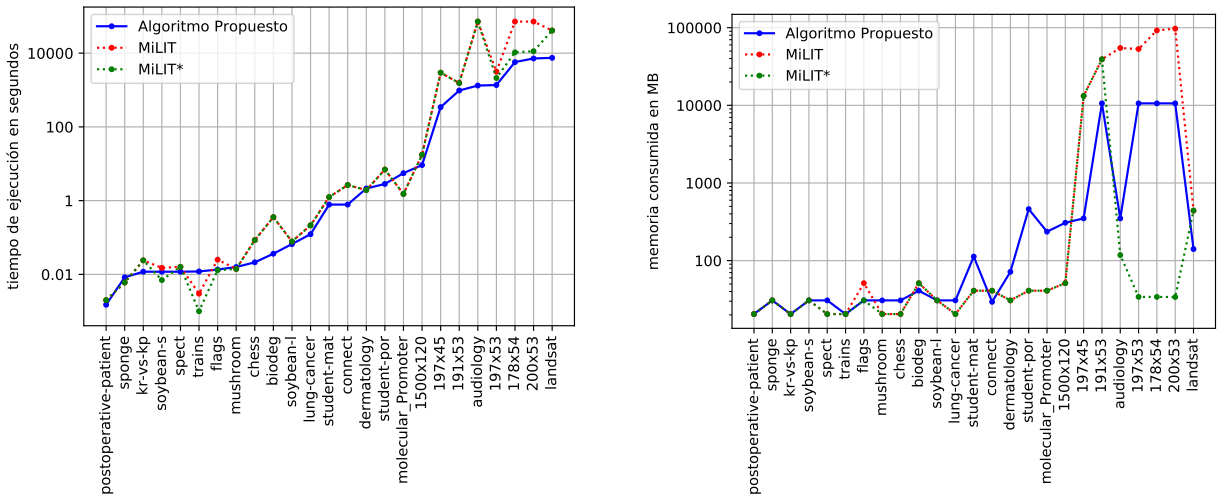
Figura 5.2: Comparación del tiempo de ejecución en segundos y los requerimientos máximos de memoria de las estrategias propuestas en [Piza-Dávila et al., 2020] y el algoritmo propuesto.



En la Figura 5.3 se pueden observar los tiempos de ejecución y el consumo de memoria, del algoritmo propuesto en esta tesis y del algoritmo *MiLIT* introducido en [Piza-Dávila et al., 2020], adicionalmente , denotado como *MiLIT**, se grafició el tiempo de ejecución y el consumo de memoria de la estrategia (*PFRC-MiLIT* o *NC-MiLIT*) de menor tiempo para las MDBS asociadas a los conjuntos de datos de la Tabla 5.1. Como se puede observar en la Figura 5.3, el algoritmo *MiLIT* falla en escoger la estrategia

adecuada en algunos casos, lo cual provoca que para las MDBS *'178x54'* y *'200x53'* no se encuentren todos los reductos más cortos antes de las 20 horas, sin embargo, de ser aplicada la estrategia correcta el algoritmo *MiLIT* hallaría todos los reductos más cortos para las MDBS *'178x54'* y *'200x53'* en 3 horas aproximadamente. No obstante, independientemente de la estrategia utilizada el algoritmo *MiLIT*, para la MDBS *'audiology'* no es capaz de encontrar todos los reductos más cortos antes de 20 horas, al contrario del algoritmo propuesto el cual es capaz de encontrar todos los reductos más cortos en 23 minutos aproximadamente.

Figura 5.3: A la izquierda se muestra el tiempo de ejecución en segundos de los algoritmos y a la derecha se muestra los requerimientos máximos de memoria en Megabytes de los mismos, en donde *MiLIT* es el algoritmo introducido en [Piza-Dávila et al., 2020] y MiLIT* es la mejor estrategia (*PFRC-MiLIT* y *NC-MiLIT*) para cada MDBS.



En la Tabla 5.4 se muestran los tiempos de ejecución del algoritmo propuesto en esta tesis y de las estrategias utilizadas en [Piza-Dávila et al., 2020], al ser aplicados a las MDBS asociadas a los conjuntos de datos de la Tabla 5.1, resaltando en color rojo el tiempo de la estrategia que escoge el algoritmo *MiLIT* para cada conjunto de datos. Los tiempos con un * significan que el algoritmo pasado ese tiempo no había sido capaz

de encontrar todos los reductos más cortos.

Tabla 5.4: Tiempo de ejecución de los algoritmos en horas, minutos y segundos, en donde PFRC-MiLIT y NC-MiLIT son las estrategias introducidas en [Piza-Dávila et al., 2020], resaltando en color rojo el tiempo de la estrategia que escoge el algoritmo *MiLIT*.

MDBS	*PFRC-MiLIT*	*NC-MiLIT*	*Algoritmo Propuesto*
postoperative-patient	00:00:00	00:00:00	00:00:00
sponge	00:00:00	00:00:00	00:00:00
kr-vs-kp	00:00:00	01:24:55	00:00:00
soybean-s	00:00:00	00:00:00	00:00:00
spect	00:00:00	00:00:00	00:00:00
trains	00:00:00	00:00:00	00:00:00
flags	00:00:00	00:00:00	00:00:00
mushroom	00:00:00	00:00:00	00:00:00
chess	00:00:00	02:03:03	00:00:00
biodeg	00:00:00	00:18:35	00:00:00
soybean-l	00:00:00	00:12:53	00:00:00
lung-cancer	00:00:00	00:00:00	00:00:00
student-mat	00:00:54	00:00:01	00:00:00
connect	00:00:02	20:00:00*	00:00:00
dermatology	00:00:02	00:00:01	00:00:02
student-por	00:02:02	00:00:06	00:00:02
molecular_Promoter	00:00:05	00:00:01	00:00:05
audiology	20:00:00*	20:00:00*	00:22:01
landsat	12:30:15	11:24:42	02:04:21
1500x120	00:00:28	00:00:17	00:00:09
197x45	00:49:13	01:58:33	00:05:43
191x53	00:25:45	00:26:49	00:16:10
197x53	00:52:32	00:35:21	00:22:45
178x54	20:00:00*	02:55:17	01:35:26
200x53	20:00:00*	03:08:04	01:59:41

Como se puede observar en la Tabla 5.4, para las matrices *'audiology'*, *'197x45'*, *'191x53'*, *'197x53'*, *'178x54'*, *'200x53'* y *'landsat'*, existe una diferencia notable entre el tiempo ejecución del algoritmo propuesto para hallar todos los reductos más cortos y el algoritmo *MiLIT* propuesto en [Piza-Dávila et al., 2020]. En la Tabla 5.5 se muestran estas matrices, donde N_{SP} es el número de subconjuntos en la partición del conjunto de atributos creada por el Algoritmo 4.2, T_{core} es el número de atributos en el core y

las columnas N_C, $DENS$, T_{RMC}, EST y N_{RMC} son las mismas columnas que en la Tabla 5.1. En la Tabla 5.5, se puede observar, que para el caso particular de la MDBS

Tabla 5.5: Conjuntos de datos en los cuales las diferencias entre el tiempo de ejecución del algoritmo propuesto y el algoritmo MiLIT son más significativas.

MDBS	N_C	N_{SP}	T_{core}	$DENS$	T_{RMC}	EST	N_{RMC}
audiology	69	54	2	0.0692	14	12	20
landsat	36	36	0	0.3445	14	7	6
197x45	45	45	0	0.2012	11	10	25
191x53	53	53	0	0.2035	10	9	9
197x53	53	53	0	0.1989	10	9	1
178x54	54	54	0	0.1931	11	9	235
200x53	53	53	0	0.2020	11	9	1056

'audiology' el número de subconjuntos en la partición generada por el Algoritmo 4.2 es de 54, esto significa, que el algoritmo propuesto inicialmente solo considera 54 atributos para determinar el tamaño real de los reductos más cortos, ver Sección 4.2, a diferencia del algoritmo *MiLIT* que considera en todo momento los 69 atributos en *'audiology'*. No obstante, si observamos las matrices *'197x45'*, *'191x53'*, *'197x53'*, *'178x54'*, *'200x53'* y *'landsat'*, se puede notar que el número de subconjuntos en la partición generada por el Algoritmo 4.2 es igual al número de atributos, que sería el peor caso para la estrategia de poda, pero, aún así, el algoritmo propuesto sigue siendo más rápido que el algoritmo MiLIT, ya que trabajar con una aproximación inicial del tamaño de los reductos más cortos reduce significativamente el espacio de búsqueda. En las MDBS *trains* y *molecular_Promoter* el algoritmo *MiLIT* es ligeramente más rápido que el algoritmo propuesto, ver Figura 5.3, esto se debe, a que, en estas matrices, el tamaño de los reductos más cortos es pequeño y el algoritmo propuesto realiza algunas operaciones adicionales para podar el espacio de búsqueda, lo cual representa un costo de tiempo adicional al de la evaluación de los candidatos a reductos. Un aspecto importante que se pudo notar en los experimentos realizados es que las estrategias propuestas en [Piza-

Dávila et al., 2020] pueden presentar un mal desempeño si la densidad de la MDBS es cercana a 0.3 y en estos casos se observa una mayor diferencia entre el tiempo de ejecución del algoritmo propuesto y el algoritmo *MiLIT*. El algoritmo propuesto ha mostrado un buen rendimiento independientemente de la densidad de la MDBS, alcanzando diferencias notables en tiempo de ejecución con respecto al algoritmo *MiLIT* para algunas MDBS con densidades cercanas a 0.3.

Al final de este experimento, se aplicó la prueba de Wilcoxon a los tiempos obtenidos por el algoritmo propuesto y el algoritmo *MiLIT* para los conjuntos de datos en la Tabla 5.1; dando como resultado que se se rechaza la hipótesis nula con un nivel de confianza del 98 %, por lo que se puede comprobar estadísticamente que el tiempo de ejecución del algoritmo propuesto es significativamente menor que el tiempo de ejecución del algoritmo *MiLIT*.

5.3 Síntesis

De los experimentos realizados se puede concluir que las predicciones realizadas por el modelo propuesto para estimar el tamaño de los reductos más cortos son satisfactorias, ya que el modelo logra una buena aproximación para las matrices de los conjuntos de datos de la Tabla 5.1, para los cuales el error de aproximación obtenido es de 3 aproximadamente, ver Figura 2. Por otra parte, partiendo de los resultados que se muestran en Figura 5.2, se puede observar que en la mayoría de los casos, el algoritmo propuesto en esta tesis calcula todos los reductos más cortos en un tiempo menor que las estrategias propuestas en [Piza-Dávila et al., 2020], sin requerir demasiada memoria, especialmente en las matrices que requieren un mayor tiempo de ejecución (*'audiology'*, *'landsat'*, *'197x45'*, *'191x53'*, *'197x53'* y *'178x54'*, *'200x53'*).

Capítulo 6

Conclusiones

Calcular todos los reductos de un conjunto de datos es un problema de complejidad exponencial, y por lo tanto, consume una gran cantidad de tiempo de cálculo. En algunos estudios se ha evaluado la factibilidad de calcular todos los reductos más cortos (reductos de menor cardinalidad), en lugar de calcularlos todos. En la literatura existen muy pocos algoritmos para calcular todos los reductos más cortos, por lo que se hace necesario el desarrollo de algoritmos eficientes que calculen todos los reductos más cortos.

En este trabajo de tesis se introdujo un algoritmo para el cálculo de todos los reductos más cortos, el cual opera sobre la matriz de discernibilidad binaria simplificada, a diferencia de otros que utilizan la función de discernibilidad Booleana. La idea principal del algoritmo propuesto en esta tesis es que si se conoce de antemano el tamaño de los reductos más cortos éstos se pueden calcular de forma eficiente. Por lo tanto, se propuso un método para estimar el tamaño de los reductos más cortos y se proponen algunas estrategias para podar el espacio de búsqueda.

Las principales estrategias de poda utilizadas por los algoritmos en la literatura que calculan todos los reductos más cortos están basadas en el concepto de contribución de un atributo a un subconjunto de atributos, no obstante evaluar dicha condición para cada subconjunto de atributos es un proceso costoso. En el algoritmo propuesto en esta tesis se introdujo un nuevo mecanismo para realizar la poda del espacio de búsqueda, el cual consiste en realizar una partición del conjunto de atributos y utilizar dicha parti-

ción para realizar la búsqueda de los reductos más cortos, esto permite podar el espacio de búsqueda, ya que muchas veces no se tienen que utilizar todos los atributos para determinar el tamaño de los reductos más cortos.

Se realizó una evaluación del algoritmo propuesto contra el algoritmo $MiLIT$ [Piza-Dávila et al., 2020], el cual es el algoritmo más reciente reportado en la literatura. Para comprobar la eficacia del algoritmo propuesto se realizaron varios experimentos en los cuales se utilizaron varios conjuntos de datos reales y algunos sintéticos. A partir de nuestros experimentos se puede concluir que el algoritmo propuesto en esta tesis es más rápido que el algoritmo $MiLIT$ en la mayoría de los conjuntos de datos evaluados, sin requerir demasiada memoria, especialmente en las matrices que requieren un mayor tiempo de ejecución y encontrando todos los reductos más cortos en tiempos razonables. También se pude concluir, que utilizar un número mayor de candidatos a reductos no mejora necesariamente el rendimiento del algoritmo y en algunos casos empeora dicho rendimiento. Por lo tanto, se puede concluir, con base en los resultados experimentales, que se han alcanzado tanto el objetivo general como los objetivos específicos de la tesis. Las contribuciones de esta tesis son:

1. Un algoritmo para obtener una estimación del tamaño de los reductos más cortos en un sistema de decisión.

2. Un algoritmo para calcular los reductos más cortos, que es significativamente más rápido que el algoritmo más reciente de la literatura [Piza-Dávila et al., 2020].

3. Una nueva estrategia de poda del espacio de búsqueda de los reductos más cortos basada en una partición del conjunto de atributos usando el concepto de subcolumna.

El objetivo general de la tesis es un algoritmo para el cálculo de los reductos más cortos y las contribuciones obtenidas en esta tesis tributan al objetivo general. Es decir, dichas contribuciones forman parte del objetivo general, pero pueden ser utilizadas de forma independiente en otros algoritmos.

6.1 Trabajo Futuro

Como alternativas para mejorar los resultados obtenidos en este trabajo se proponen las siguientes lineas de trabajo futuro:

1. Mejorar la estimación del tamaño de los reductos más cortos, ya que cuanto más cercana a la óptima sea la predicción del tamaño de los reductos más cortos menos tiempo será utilizado en buscar el tamaño real de los reductos más cortos y por ende mejoraría la velocidad del algoritmo.

2. Estimar el número óptimo de candidatos a generar, ya que en algunos conjuntos de datos es mejor generar más candidatos a reductos que en otros, porque si bien el número de atributos en el conjunto de datos influye directamente en el número de candidatos a generar, existen otras características que pueden utilizarse para seleccionar este valor.

3. Buscar algunas condiciones que permitan eliminar aún más candidatos a reductos en el proceso de la búsqueda del tamaño de los reductos más cortos, de forma tal, que al buscar todos los reductos más cortos, el número de candidatos a evaluar sea menor.

4. También, se planea desarrollar una implementación paralela del algoritmo propuesto que se pueda ejecutar tanto en GPU como en CPU para así lograr obtener un menor

tiempo de ejecución. Para esto, es recomendable utilizar TensorFlow para dicha implementación, ya dicha biblioteca posee gran número de funciones para realizar operaciones con matrices y permite implementar el cálculo a una o más CPU o GPU en equipos de escritorio o servidores.

Bibliografía

Alganza, Y. S. and Porrata, A. P. (2003). LEX: Un nuevo algoritmo para el cálculo de los Testores Típicos. *Ciencias Matemáticas*, 21(1):85–95.

Alwesabi, K., Gui, W., Yang, C., and Rajeh, H. (2016). Feature Selection based on Rough Sets and Minimal Attribute Reduction Algorithm. *International Journal of Hybrid Information Technology*, 9:333–346.

An, A., Huang, Y., Huang, X., and Cercone, N. (2004). Feature selection with rough sets for web page classification. In *Transactions on Rough Sets II*, pages 1–13. Springer.

Bazan, J., Nguyen, H., Nguyen, S., Synak, P., and Wroblewski, J. (2000). Rough set algorithms in classification problems, Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. *Physica-Verlag, Heidelberg, Germany*, pages 49–88.

Carreira-Perpinán, M. A. (1997). A review of dimension reduction techniques. *Department of Computer Science. University of Sheffield. Tech. Rep. CS-96-09*, 9:1–69.

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.

Chouchoulas, A. and Shen, Q. (2001). Rough set-aided keyword reduction for text categorization. *Applied Artificial Intelligence*, 15(9):843–873.

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

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Granados, R. M. (2016). Modelos de regresión lineal múltiple. *Granada, España: Departamento de Economía Aplicada, Universidad de Granada*.

Grzymala-Busse, J. W. (2005). Rough set theory with applications to data mining. In *Real World Applications of Computational Intelligence*, pages 221–244. Springer.

Hu, X. (1995). *Knowledge discovery in databases: an attribute-oriented rough set approach*. PhD thesis, University of Regina Regina, Canada.

Jensen, R. and Shen, Q. (2003). Finding rough set reducts with ant colony optimization. In *Proceedings of the 2003 UK workshop on computational intelligence*, volume 1, pages 15–22.

Jensen, R. and Shen, Q. (2008). *Computational intelligence and feature selection: rough and fuzzy approaches*, volume 8. John Wiley & Sons.

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

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.

Jiri Sedlacek, T. H. (2017). VisualVM [http://visualvm.java.net/].

Lazo-Cortés, M. S., Martínez-Trinidad, J. F., and Carrasco-Ochoa, J. A. (2018). Class-specific reducts vs. classic reducts in a rule-based classifier: a case study. In *Mexican Conference on Pattern Recognition*, pages 23–30. Springer.

Lias-Rodríguez, A. and Pons-Porrata, A. (2009). BR: A new method for computing all typical testors. In *Iberoamerican Congress on Pattern Recognition*, pages 433–440. Springer.

Lias-Rodriguez, 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.

Lin, T. Y. and Yin, P. (2004). Heuristically fast finding of the shortest reducts. In *International Conference on Rough Sets and Current Trends in Computing*, pages 465–470. Springer.

Mughal, K. A. and Rasmussen, R. W. (2003). *A programmer's guide to Java certification: a comprehensive primer*. Addison-Wesley Professional.

Nguyen, H. S. and Skowron, A. (1997). Boolean reasoning for feature extraction problems. In *International Symposium on Methodologies for Intelligent Systems*, pages 117–126. Springer.

Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2005). Abstract DPLL and abstract DPLL modulo theories. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 36–50. Springer.

Øhrn, A. (2000). Discernibility and Rough sets in Medicine: Tools and Applications; PhD Thesis, Norwegian University of Science and Technology, Department of Computer and Information Science: Trondheim, Norway.

Pawlak, Z. (1982). Rough sets. *International journal of computer & information sciences*, 11(5):341–356.

Piza-Dávila, I., Sánchez-Díaz, G., Lazo-Cortés, M. S., and Villalón-Turrubiates, I. (2020). An Algorithm for Computing Minimum-Length Irreducible Testors. *IEEE Access*, 8:56312–56320.

Rajalakshmi, A., Vinodhini, R., and Bibi, K. F. (2016). Data Discretization Technique Using WEKA Tool. *International Journal of Science, Engineering and Computer Technology*, 6(8):293–298.

Rauszer, C. and Skowron, A. (1992). The discernibility matrices and functions in information systems. *Intelligent Decision Support-Handbook of Applications and Advances of the Rough Sets Theory, Knowledge Engineering and Problem Solving*, 11:331–362.

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.

Saha, S., Murthy, C., and Pal, S. K. (2007). Rough set based ensemble classifier forweb page classification. *Fundamenta Informaticae*, 76(1-2):171–187.

Sanchez-Díaz, G. and Lazo-Cortés, M. (2007). CT-EXT: An Algorithm for Computing Typical Testor Set. In Rueda, L., Mery, D., and Kittler, J., editors, *Progress in Pattern Recognition, Image Analysis and Applications*, pages 506–514, Berlin, Heidelberg. Springer Berlin Heidelberg.

Siedlecki, W. and Sklansky, J. (1993). On automatic feature selection. In *Handbook of Pattern Recognition and Computer Vision*, pages 63–87. World Scientific.

Sil, J. and Das, A. K. (2012). Variable Length Reduct Vs. Minimum Length Reduct-A Comparative study. *Procedia Technology*, 4:58 – 68. 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.

Singh, S. and Prasad, B. (2008). User relevance feedback analysis in text information retrieval: A rough set approach. In *Soft Computing Applications in Business*, pages 147–178. Springer.

Starzyk, J., Nelson, D., and Sturtz, K. (2000). A Mathematical Foundation for Improved

Reduct Generation in Information Systems. *Knowledge and Information Systems*, 2:131–146.

Suraj, Z. (2004). An introduction to rough set theory and its applications. *ICENCO, Cairo, Egypt*, 3:80.

Susmaga, R. (1998). Computation of shortest reducts. *Foundations of Computing and Decision Sciences*, Vol. 23, No. 2:119–137.

Sánchez-Díaz, G., Piza-Dávila, 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 *Mexican International Conference on Artificial Intelligence*, pages 92–103. Springer.

Wang, G. Y., Zhao, J., An, J. J., and Wu, Y. (2004). Theoretical study on attribute reduction of rough set theory: comparison of algebra and information views. In *Proceedings of the Third IEEE International Conference on Cognitive Informatics, 2004.*, pages 148–155. IEEE.

Wang, J. and Ju, W. (2001). Reduction Algorithms Based on Discernibility Matrix: The Ordered Attributes Method. *J. Comput. Sci. Technol.*, 16:489–504.

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.

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.

Wroblewski, J. (1995). Finding minimal reducts using genetic algorithms. In *Proccedings of the second annual join conference on infromation science*, volume 2, pages 186–189.

Yang, P., Li, J., and Huang, Y. (2008). An attribute reduction algorithm by rough set based on binary discernibility matrix. In *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, volume 2, pages 276–280. IEEE.

Yang, P., Li, J., and Huang, Y. (2008). An Attribute Reduction Algorithm by Rough Set Based on Binary Discernibility Matrix. In *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, volume 2, pages 276–280.

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

Yao, Y., Zhao, Y., and Wang, J. (2008). On reduct construction algorithms. In *Transactions on computational science II*, pages 100–117. Springer.

Ye, D. Y. and Chen, Z. J. (2002). A new discernibility matrix and the computation of a core. *Acta electronica sinica*, 30(7):1086–1088.

Yekkala, I. and Dixit, S. (2018). Prediction of heart disease using random forest and rough set based feature selection. *International Journal of Big Data and Analytics in Healthcare (IJBDAH)*, 3(1):1–12.

Yu, L., Wang, S., and Lai, K. K. (2009). A Rough-Set-Refined Text Mining Approach for Crude Oil Market Tendency Forecasting. 2.

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.

Zhou, J., Miao, D., Feng, Q., and Sun, L. (2009). Research on complete algorithms for minimal attribute reduction. In *International Conference on Rough Sets and Knowledge Technology*, pages 152–159. Springer.